UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

TRABAJO DE FIN DE GRADO

Diseño y Desarrollo de un Prototipo para Sistemas de Control en el Contexto de la Internet de las Cosas

GRADO EN INGENIERÍA EN
TECNOLOGÍAS DE TELECOMUNICACIÓN

AUTOR: ALEJANDRO ESCUDERO FERNÁNDEZ

TUTOR: Dr. DANIEL DÍAZ SANCHEZ

Leganés, Octubre de 2015

Quiero dar las gracias a todas aquellas personas que han estado a mi lado apoyándome tanto en los buenos como en los malos momentos y que sin ellas no hubiera sido capaz de llegar hasta aquí, incluyendo a quienes ya no estáis aquí para poder verlo. Siempre estaréis en mi corazón.

ÍNDICE GENERAL DEL TRABAJO

ÍNDICE DE FIGURAS

ÍNDICE DE DATOS Y TABLAS

ÍNDICE DE EXPRESIONES

I. INTRODUCTION

I.1 INTRODUCTION TO THE INTERNET OF THINGS

During the last two decades, we have seen how the use of devices connected through the Internet has increased exponentially [1], since the launch of desktop computers and laptops until the current technologies such us smartphones and tablets, which have become the truly protagonists.

In fact, if we focus on smartphones and tablets, we can see how nowadays society is developing a dependence/addiction to this kind of devices that is increasing more and more for different uses such us communication, entertainment or several tasks in different environments of our lifes thanks to the applications like interacting with your bank, having notifications about the weather or the news around the world, shopping online or controlling other devices remotely. All of these facts, that wouldn't even haven been imaginable just ten years ago, have become a reality.

In the graphic below provided by Cisco Systems we can check the growth of the number of devices conneted to the Internet since 1992 until 2020 [2].

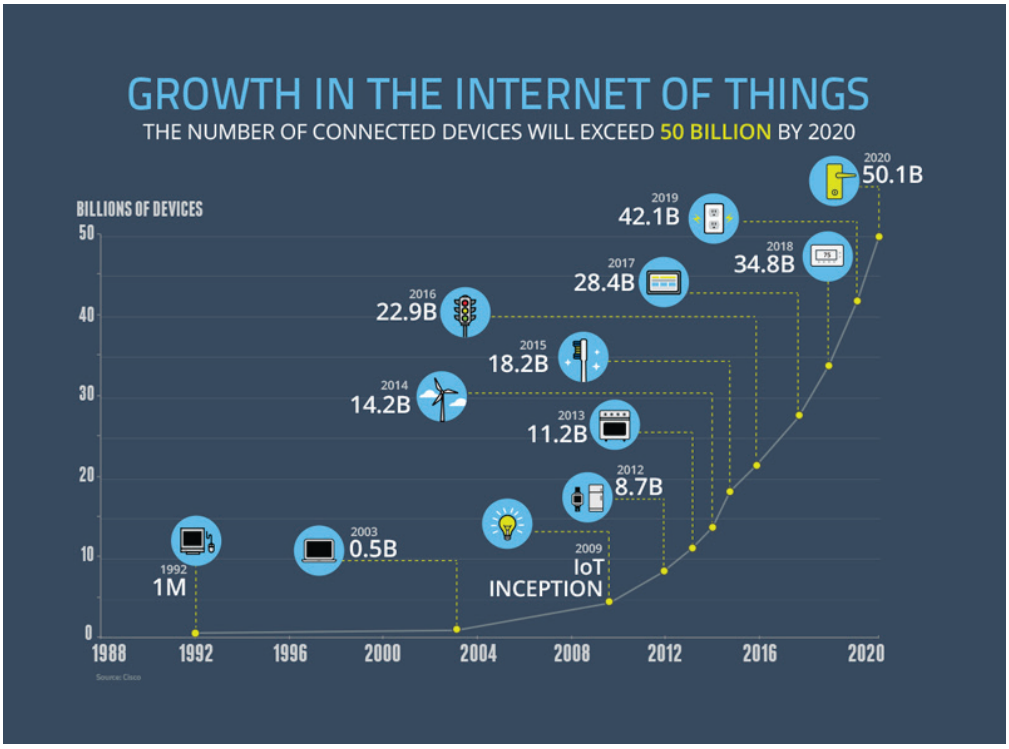


Figure 1. "Growth in the IoT" graph showing connected devices every year. ***[1]***

As it can be seen in the previous graph, people will tend to the the dependence on the Internet on their lifes; the dependence on getting connected to the Internet

for one or several purposes. This is known as the 'Internet of Things', from now on 'IoT'.

In this project we will analize the possibilities and consequences of the communication between objects and mobile devices through the Internet, which goal is controlling and getting data remotely from other objects in order to process/control these objects, including its impact in terms of security systems and messages.

I.2 MOTIVATION

Nowadays, it's usual the interaction between humans and objects or electric systems by switches and/or remote control systems based on infrared such as remote control or speech recognition.

In addition, some multiple complex systems can interact between them. One example of this fact could be the heating systems or the air conditioning that interact with a temperature sensor. However, these systems are previously designed by the maker of the product, that is to say that they can't be considered an IoT system since they are isolated systems and with no connection to the Internet; they can't have any improvement except some firmware updates on the product.

Furthermore, this tendency to the IoT on devices and cloud services (i.e. Cloud Sensing, or FlowCloud [3]) makes necessary additional security measures that current systems don't have. These measures are advisable in order to avoid that hackers can, not only get personal data from the users, but also that they can use this information for unwanted purposes.

So, we could increase the user's confidence by adding an extra level of security in order to keep safe the data received/send by users through the Internet because this data could travel among other external networks or in some Cloud that could not be reliable. This is because we can't control the systems that the data packets go through.

I.3 GOALS

The purpose of this Bachelor Thesis is to propose a study case that consists in a simulation of interconnected nodes whose characteristics might be similar to the future Internet of Things systems with the extra layer of security encryption proposed responsible in sensible data protection, providing encryption, re-encryption and decryption of all data packets used in the transmission. This case shows how all security elements and nodes are responding to the applied algorithms, testing some different levels of encryption and different messages size in a concrete instant of time.

In the future, it could be ideal some standardization by an organization i.e. ITU or ISO in order to be able to have some established networks and security protocols, algorithms and a minimum requirements in order to stablish a better quality of products whoever its manufacturer is.

With these regulations, some possible technical incompatibilities between products made by different manufacturers would disappear, like the cheaper production and distribution costs and diversity of them, etc.

In this thesis, the complexity of the Internet of Things and the great difficulty to solve all the problems that will occur during its implementation is known, because the ellaboration of this thesis is limited on time and human resources. There is just one person is involved in this thesis: its author.

Here there is proposed one possible solution about one of the most important problems that usually people think about: Are my personal data and their integrated security traversing external networks? Can someone intercept them and know everything about me?

The proposed solution will be shown in this thesis as a simulation of an IoT or Cloud Sensing environment using a concrete actor model that is provided by the Akka.io Java Library and integrated with a Proxy Re-Encryption System in order to allow sensible or personal data to travel around external networks or Clouds with an extra layer of security.

I.4 CONTENTS OF THIS REPORT

This thesis is divided in three parts: Indexes and Mandatory Sections in English Language, Thesis Body and its annexes, and the Bibliography.

The Thesis Body is in Spanish and it is divided in several sections differentiated as follows:

- In Chapter 1 we start the thesis with an introduction, motivation, objectives and showing a summary to its content.

- In Chapter 2 we will be talking about the State of Art explaining the source of the problem and defining the different technologies needed in order to apply the proposed solution that will be explained later.

- In Chapter 3 we will explain the design and development of the main solution, show the environment where we will make some simulation and hardware perfomance testing and show what problems come up during the process.

- In Chapter 4 we will show all significant testing results, commenting all of them and recommend or not the security levels applied in each of them.

- In the end of this document, we includ some annexes about the planning, budget and actual economic framework applied the problem shown in this project.

II. EXTENDED ABSTRACT

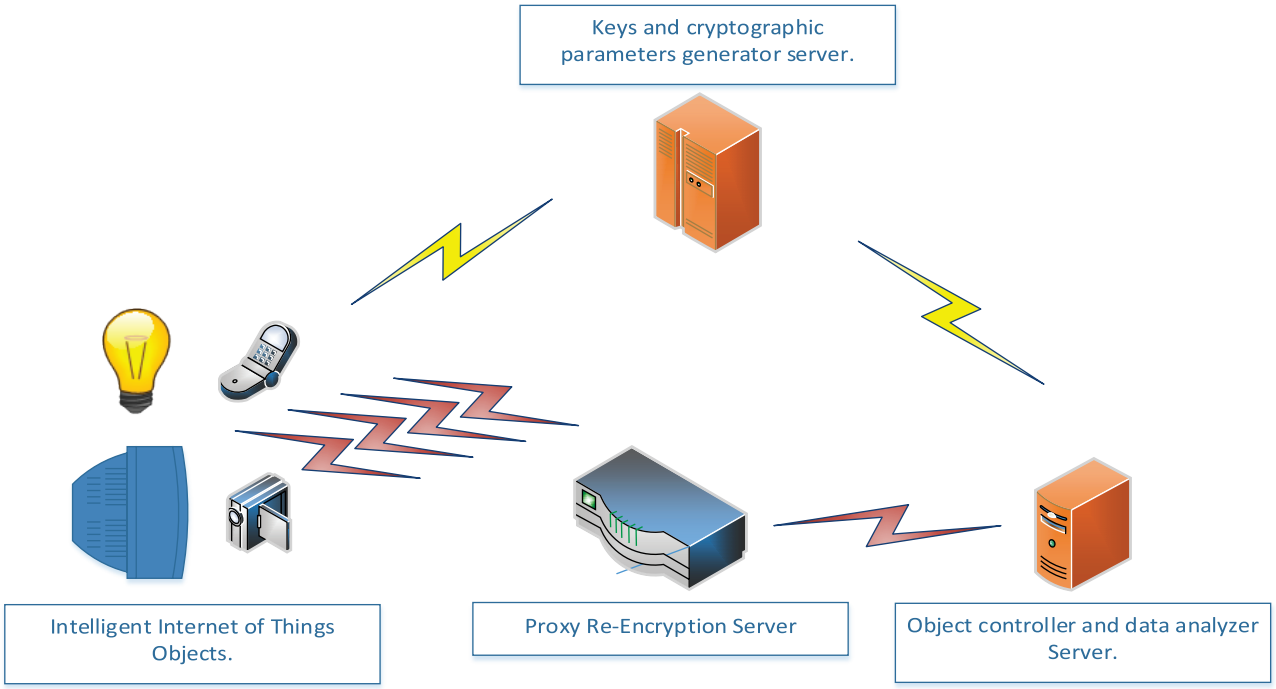
The purpose of this Bachelor Thesis is to propose a possible solution to the problem involving the delivery of sensitive information through a mobile device connected to a network. Said network might not be reliable, because mobile devices usually stablish a wireless connection to the Network at different access points.

The aforementioned idea might be supported in a future environment of the 'Internet of Things' such as Crowd Sensing. This model shows a group of users sharing private information like their GPS location and/or their average speed in that moment so that all this information is processed in a server and shared anonymously by all the users of the application with a view to obtain live data about the traffic in a road at a certain time, among other functions.

Internet of Things is defined as the moment of humankind's technological evolution when there will exist a larger number of completely automatical devices —able to take their own decisions— than personal devices controlled by a person. The medium-long term trend suggests that there is an exponential progress in this principle because eventually more and more devices work automatically.

Thus, here there is a proposal of the design and development of an environment related to the Internet of Things where the following elements should become available:

- One or more intelligent elements, from intelligent temperature sensors to mobile devices that send sensitive information.
- A keys and cryptographic parameters generator server.
- A central controller server that will be the destination point of the data.
- A proxy re-encryption server that recodes the messages by using the AFGH algorithm based on a Proxy Re-Encryption.



Final testing topoligy designed in this Bachelor Thesis

A scenario including the Proxy Re-encryption methodology might be applicable to those cases when a certain degree of privacy is required during the constant exchange of keys between the origin and the destination from one or several transmissions of sensitive or private data, even if there is some mistrust about the data safety in a local area network.

This is the reason why a third element considered as a re-encryption proxy that serves as an intermediary. It cannot even decode the messages that traverse it because it does not own any of the private key to decode the message. It just has a re-encryption key between the ends that —depending on the algorithm that has been used— may be unidirectional or bidirectional.

In case it is a new unidirectional re-encryption key, it is required that the proxy has two re-encryption keys for each connection between the ends if they are aimed to establish a mutual communication. However, in case it is a bidirectional re-encryption key, just one key would be required for each communication between the ends regardless of the direction of the transmission of the messages in that moment.

This kind of algorithms provides an additional protection layer to the one that already exists in the network protocol or application, assuring a greater warranty of confidentiality in the transmission of sensitive data through a channel of transmission which is not controlled directly because it does not belong to a private

owner or because a company of the competition might be interested in obtaining some confidential data.

The AFGH algorithm —included in NiCS and JPBC libraries— is based on the cryptosystematic way of ElGamal and introduces the concept of Re-Encryption Key. The application of said key to a coded message enables it to be decoded with the private key of the addressee of the message. The proxy is not able to know the message or any of the two private keys of the ends. Furthermore, the proxy will not be able to calculate the re-encryption key in one direction of the communication channel in case it receives the re-encryption key generated by one of the ends for the other direction. As a consequence, this algorithm might also be useful in environments where the communication is established in just one direction.

With this aim, each and every one of the aforementioned elements of the IoT must be modelled on the actor model, so that every object can be an actor performing the tasks that this object would perform in the real world.

The actor model is a concept introduced by Carl Hewitt in 1973. It came up because the concurrent computing in distributed environments was too complex. In said model —where basic primitives of computing are actors— each actor represents an object with a mailbox of asynchronous messages waiting to be answered in a specific way according to its content.

As a consequence, the use of actors consists on the use of concurrent threads being executed in parallel. Each and every one of them can communicate among them asynchronously and create more actors hierarchically multiplying its capacity of action or with new utilities in case these are needed.

One of the most important libraries used by the actor model is Akka in its version for Java. It is a programming language aimed at objects. Its main feature is that it may be executed in any system containing the Java virtual machine in the device. This virtual machine can be installed in any kind of computer, independently of its operating system (compatible with Windows, Mac OS, Linux, etc.) and even in other specific devices with the ability to execute it.

Akka is an Open Source library for Java and Scala (another programming language aimed at objects) with the purpose of making the most of the potential of the actor modeling in order to make easier the development of a complex application that requires the concurrent execution of tasks, tolerant of constant changes and scalable, in a friendly way and based on the reactive systems model. On Akka's website, all the necessary information to start developing simple concurrent

applications is at everyone's disposal. It simulates perfectly a distributed complex environment, even if it is executed in just one machine with simulation purposes.

For the delivery and reception of messages between actors on Akka for Java, the use of an objects serializer is required.

A serializer in Java is a system able to transform into bytes any object as it would be done in a real network environment when transmitting said object, besides restoring it in the destination actor. Java has its own serializer by default. However, it has the inconvenient that it is necessary to add the 'Serializable' interface to each and every one of the classes liable to be sent at a certain moment.

This is the reason why in this Bachelor Thesis will be used for Akka an external serializer able to serialize any kind of class independently of whether it is from the user or from another library, such as the classes related to the library JPBC or AFGHGlobalParameters —the last one is part of the NiCS library—.

As to the development of the simulation environment, the support is a virtual machine formatted with Windows 10 Pro, 3 GB of RAM, a Solid State Drive, and the dedication of two logical core of a processor i5-2400S at 2.49 GHz. The version used is the 1.8.0_40 for the Java virtual machine and the JDK (Java Development Kit) in the Windows 10 Pro version. The 8.0.2 version for Netbeans —compatible with Java SE and EE— will be used as the development environment.

Once the development has been finished, different tests have been made in order to check the proper functioning of each and every one of the actors. In addition, some efficiency tests will be made for some configurations of the AFGH algorithm related to the key size and the key strength.

The following efficiency tests have been made for three kinds of AFGH algorithm strength:

- Calculation of the average times for generating the cryptographic parameters and the generator numbers *'g'* and *'Z'*.
- Calculation of the average times for generating a pair of keys (a public and a private key).
- Calculation of the average times for processing and sending a message from the origin until the destination.

The different algorithm strengths are shown in the following chart:

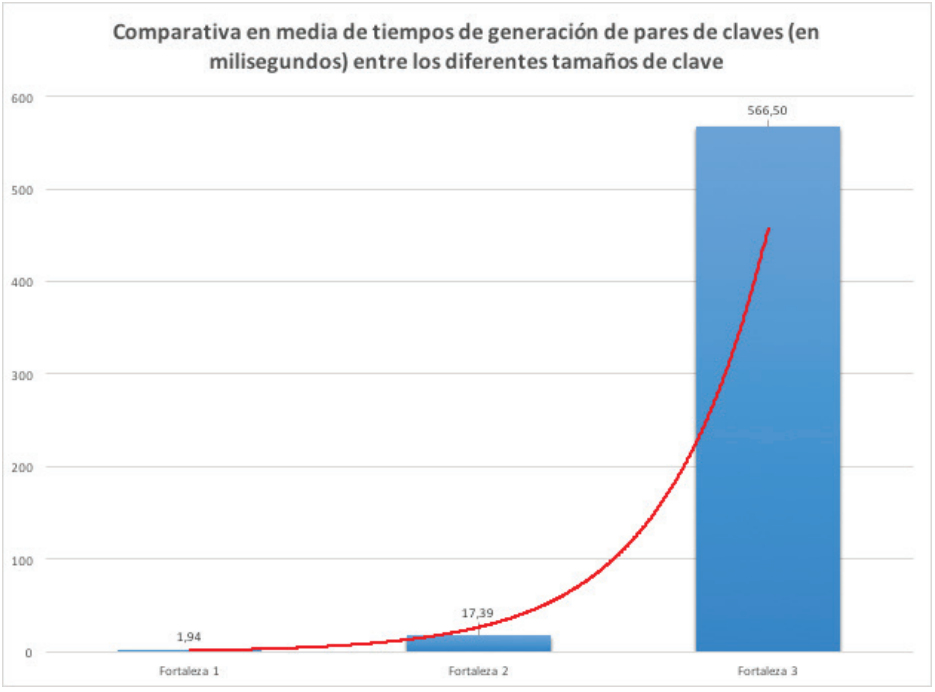
Bits of number 'q'	Bits of number 'r'
512 bits	**160 bits**
1536 bits	**256 bits**
7680 bits	**512 bits**

Once the test have been made in the actors in order to obtain the values of the time needed to generate a pair of public and private keys in .csv format, the average values about the generation times of the pair of keys have been the following:

$t_1 = 1{,}94$ ms	*for strength 1st:*	$q = 512$ *bits*	$r = 160$ *bits*
$t_2 = 17{,}39$ ms	*for strength 2nd:*	$q = 1536$ *bits*	$r = 256$ *bits*
$t_3 = 566{,}50$ ms	*for strength 3rd:*	$q = 7680$ *bits*	$r = 512$ *bits*

Thus, according to the desired security in the communication, it would be possible to choose any of the three, because all the values are within a coherent waiting time. However, it is necessary to analyze the security-speed commitment, because in some environments where there is a possibility to change the generator parameters regularly, the Strength 1st would be enough. Whereas, for average security systems, the ideal might be the use of AFGH algorithm with the configuration of the Strength of 2nd keys.

As it has been mentioned, a larger size of parameters —and therefore, of keys, the runtime grows exponentially, as it can be seen:



Key-pair generation time comparative (in miliseconds).

The growing exponential trend regarding the generation time and calculation of any cryptographic action is demonstrated. The larger the numbers are, the longer the time will be needed.

This is why in normal conditions the use of the Strength 2^{nd} is the more recommended to be used, because it has the best security-runtime relation. However, in some sceneries where parameters are generated the keys are renewed constantly so the Strength 1^{st} of the AFGH algorithm might also be used.

III. CONCLUSIONS

III.1 CONCLUSIONS

Once the thesis has been finished, it is verified that the system works according to the expected functionalities and results basing this verification on the analysis of each one of the possible cryptographic solutions proposed in the design and the analysis of the processing times of each one of them.

After some tests of the delivery and reception of messages with different encoded content, the results shows that it achieves the purpose fixed at the beginning. This purpose was about maintaining a secure communication with a point-to-point code with the use of an intermediary (proxy), regardless of the networks crossed by the data that might not be reliable.

As to the results of the previous tests and validations, these have been the final values (for the time):

PRUEBAS	Strength 1st	Strength 2nd	Strength 3rd
Parameter Generation	247,27 ms	5,40 seg	37,54 min
Key Pair Generation	1,94 ms	17,39 ms	566,50 ms
Processing, deliver and receive messages	113,86 ms	665,05 ms	23,70 seg

The data analysis provided by the actors in .csv format, the conclusion is that the recommended strength of the AFGH algorithm for this scenario is the 'Strength 2', whose size is *'q' = 1536 bits* and *'r' = 512 bits*, because it has the best security-runtime.

Even if for some cases a big amount of simultaneously and constantly processed data is required, if the keys are renewed periodically there would not be any inconvenience in using the security level provided by the 'Strength 1' configuration. Anyway, with occasional exceptions when information is sent seldom but it is very sensitive, the third degree of strength of the algorithm may be used.

The most significant difficulties in this Bachelor Thesis have been related to the union of the different technologies used thanks to the libraries that have been referred to in 3.3. Desarrollo de la Solución (in Spanish).

Originally, the generation of the parameters was made exclusively in the actor in charge of the parameters generation. However, there was the inconvenient regarding the process of coding and decoding —which should have been done in said actor—, so a change in the design of the petition/generation of keys had to be made. At the time when the pair of keys is sent to the actor who requests it, also a chain with values determining the generated parameters is included. Thus, they can be 'regenerated' in the rest of the actors without the needed of calculating again the numbers *'q'* and *'r'*. This is necessary because the generators *'g'* and *'Z'* are necessary when coding, recoding and decoding.

Further on, another problem came up. Initially, the Java's serializer by default was being used, but it is not able to serialize classes that have already been precompiled. This is why the serializer Kryo started to be used.

Lastly, the serializer Kryo tended to go into an infinite loop when trying to serialize an AFGHGlobalParameters instance and those related to the Tuple object. This is the reason why the data about the parameters have been sent as a chain along with the actors' pair of keys. For the first case, the reconstruction at the destination of the parameters had to be made. For the second one, the separation of the elements composing the Tuple (in Array format) in several independent elements had to be made because for this thesis they are always composed by three elements. Thus, this new object named 'ElementosTuple' —that can be serialized in Kryo— was created.

This project has motivated me to know in further detail the processes traversed by the messages while these are suffering different coding processes. I also had the chance to work with a development environment like Netbeans IDE —which I had never used before— and the union of different functionalities in the same project working with other people's imported Java libraries. And last but not least, I have experienced the ease of making complex, asynchronous systems with wait thanks to the actor modeling that the Akka library has in it.

III.2 FURTHER WORK

Regarding the future lines about this Project, I consider that it has a big headroom for growth. In this section, some of the possible options to ameliorate this scenario are proposed.

A way more complex system could be designed, including some temperature sensors, elements dependent of other bigger systems, or even the simulation of mobile devices sending and receiving information from a real Crowd Sensing, connecting the actor "Controlador de Objetos" to a new actor in charge of managing a MySQL data base so that said mobile devices can send petitions for data contribution or reception for those already existing in the data base.

Another improvement that could be made is the implementation of other cryptographic algorithms aimed to make some tests in them in order to verify its efficacy in a actor model that simulates elements of the IoT. For example, the implementation of the 'BBS' algorithm —which is mentioned in the section 2.5 Proxy Re-Encryption— or other more complex based in the same philosophy. This way, a larger number of results might be obtained to determinate which one is more useful in this environment.

As to the possible modifications in the technologies that has been used, the use of the Java's serializer by default could have not been used. This way, the only one used for the project would have been the one imported by the Kryo library for Akka, because the scalability of this one cannot be compared to the one provided by the Java's serializer by default; especially the possibility of serializing almost any object.

CAPÍTULO 1: INTRODUCCIÓN

1.1 INTRODUCCIÓN

Durante la últimas dos décadas hemos podido comprobar cómo el uso de dispositivos con conectividad a Internet se ha incrementado exponencialmente [1] desde la presencia de los equipos de sobremesa y portátiles en los hogares hasta la llegada de los smartphones y tabletas, los cuales son cada vez más avanzados.

De hecho, si nos centramos en los smartphones y tabletas, podemos afirmar que la sociedad cada vez tiene más dependencia y necesidad sobre este tipo de dispositivos; los usuarios pueden utilizar dichos dispositivos para usos de comunicación, entretenimiento o para la realización de diversas tareas con el entorno gracias a las aplicaciones disponibles para cada plataforma. Dichas 'apps' permiten al usuario interactuar con sus cuentas bancarias, recibir notificaciones sobre noticias/información del tiempo, pedir la compra a domicilio o poder controlar desde el dispositivo móvil algún equipo o dispositivo remotamente; esto es algo que hace tan solo diez años podía considerarse como una idea futurista se ha hecho realidad.

En la siguiente gráfica proporcionada por Cisco Systems podemos observar la evolución en cuanto a tipo y cantidad de dispositivos conectados a Internet desde el año 1992 hasta 2020. [2]

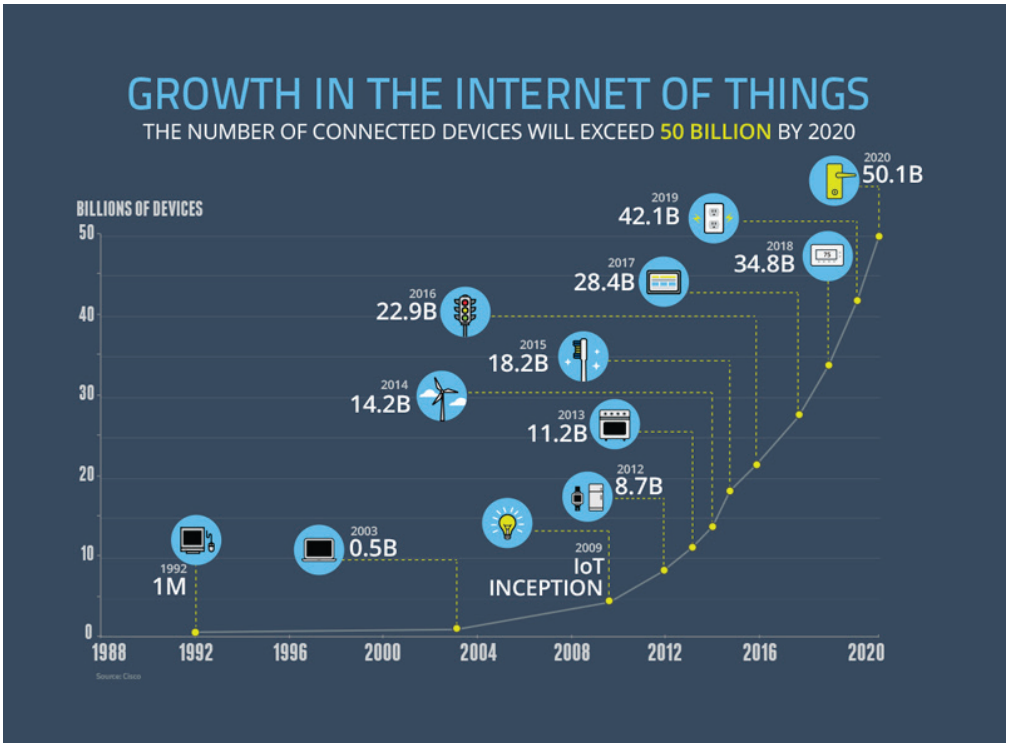


Figura 2. "Auge de la Internet de las Cosas" gráfica mostrando el incremento de dispositivos interconectados ***[1]***

Se puede observar que se tiende a la conexión de cualquier tipo de objeto de uso cotidiano a Internet, es decir, en un futuro cercano tecnológicamente hablando existirán más sistemas automáticos conectados a Internet que dispositivos personales, a éste fenómeno se le conoce como "Internet of Things" o en castellano la "Internet de las Cosas", en adelante denotado con las siglas 'IoT'.

En este proyecto vamos a analizar las posibilidades y las consecuencias de realizar dicho proceso de interconexión entre objetos y dispositivos móviles por Internet, cuya finalidad sea la de controlar o adquirir datos de objetos remotamente para su posterior procesado o control de los mismos, incluyendo su impacto en cuanto a seguridad de los sistemas y sus mensajes intercambiados.

1.2 MOTIVACIÓN

En la actualidad, es habitual interactuar con objetos y sistemas eléctricos mediante interruptores y/o sistemas de control remoto basado en luz infrarroja, véase mandos a distancia o incluso realización de acciones mediante reconocimiento de voz o imágenes.

Es posible encontrar en la actualidad dispositivos que interactúan en conjunto, como puede ser un sensor de temperatura que active o desactive un sistema de calefacción o aire acondicionado. Dichos sistemas están aislados unos de otros y sin ningún tipo de conectividad a Internet, su utilidad es especificada a priori según el diseño del fabricante de dicho producto y realiza siempre las mismas acciones en un entorno cerrado. Es necesaria la presencia de un humano para regular continuamente sus acciones de funcionamiento, razón por la que no son sistemas autónomos.

La tendencia hacia la Internet of Things con la integración de los dispositivos con Internet y los servicios en "la nube" como, por ejemplo, los sistemas "Cloud Sensing" o sistemas como FlowCloud [3], hacen necesarias unas medidas de seguridad adicionales que los sistemas en la actualidad no tienen. Dichas medidas serían recomendadas para evitar la captura de datos sensibles o de carácter personal por un atacante malicioso, ya que podría realizar acciones indeseadas con dichos datos.

Por ello, agregando una capa adicional de seguridad compatible con un entorno de comunicación asíncrono nos dotará de una mayor confianza sobre la integridad y la intimidad de los datos enviados y recibidos entre los dispositivos de los usuarios a través de la Red. Dichos datos podrían viajar a través las mismas y/o sistemas Cloud que no tienen por qué ser confiables; no tenemos por qué disponer del control sobre las redes por los que viajan los paquetes de datos que contienen nuestra

información y hace necesaria una implementación de seguridad extremo a extremo.

Aunque IPSEC o TLS son sistemas de seguridad extremo a extremo, pero no son de aplicación para un entorno de este tipo. En un entorno de Crowd Sensing pueden llegar a estar enviando muestras o paquetes millones de elementos, siendo el coste de comunicar todos ellos de forma síncrona y en tiempo real enorme.

Por otro lado, no todos los elementos necesitan disponer de la información en tiempo real, sino que se almacena y se procesa más adelante, por ello, es posible almacenar datos en lugares intermedios en función del estado de la red; de realizar operaciones como "carry and forward", es decir, guardar en algún lugar intermedio para reenviar al destino en el momento óptimo o necesario. Por este motivo surge la necesidad de disponer de entidades intermedias como los proxies.

1.3 OBJETIVOS

La finalidad de este Trabajo de Fin de Grado es la de proponer un caso de estudio relacionado con un escenario que simula un conjunto de nodos interconectados cuyas características serían similares a las que podría tener a largo plazo un sistema de control y seguridad que proteje información sensible como puede ser un entorno de Cloud Sensing proporcionando medidas de cifrado, recifrado y descifrado de paquetes de datos a lo largo de la transmisión de los mismos de forma asíncrona. De esta forma, podremos observar cómo se comportan todos los elementos que lo componen en función del nivel de seguridad aplicado y dependiendo del tipo de información que deseen enviarse en un instante de tiempo determinado.

Lo ideal en un futuro a medio-largo plazo, en caso de continuar favorablemente la tendencia en auge sobre la implementación de los sistemas de la Internet of Things, es que existiera una estandarización por una agencia reguladora de normalización como por ejemplo la ITU o la ISO de los protocolos utilizados, de los algoritmos de seguridad utilizados y de unos requisitos mínimos de calidad en todos los productos independientemente del fabricante de los mismos.

De producirse dicha estandarización contribuiría a evitar incompatibilidades técnicas y fomentaría el abaratamiento en cuanto a los costes de producción y distribución de dichos productos; además, podría existir interoperabilidad entre productos similares de diferentes compañías gracias a que los productos deberían cumplir una serie de obligaciones en el diseño de todos ellos.

Supongamos que en un momento dado, un dispositivo móvil está conectado a un servidor, el cual dispone de unos datos de carácter sensible. De repente, dicho dispositivo móvil cambia de ubicación y por ello establece una conexión inalámbrica

a una red menos confiable.

Para este caso se podría proponer el uso de mecanismos que permitan inyectar un mensaje en dicha red de carácter no confiable y, una vez en la red de destino que ya sí es confiable, realizar las operaciones necesarias para deshacer el cifrado que fue aplicado en el dispositivo móvil de origen.

En este trabajo se desea abordar uno de los problemas que es la preocupación de la sociedad por la seguridad de sus datos y la integridad de los mismos al atravesar redes no confiables, proponiendo una posible solución a dicho problema, siendo la simulación de un entorno de IoT o Cloud Sensing con el modelo de actores que proporciona Akka.io e integrarlo con un sistema de Proxy Re-Encryption para permitir que el envío de información sensible o de carácter personal sea seguro al establecer una comunicación con una red o Cloud que no sea de confianza o no se disponga del control de la misma, ya que un dispositivo móvil puede conectarse a redes inalámbricas diferentes en función de la localización del mismo y, por ello, dichas redes pueden ser no confiables.

Dicha solución debe servir para poder cifrar todos los mensajes con un tiempo medio de procesamiento aceptable y de forma estable, teniendo en cuenta que la fortaleza del algoritmo debe ser lo suficientemente buena para evitar un ataque por fuerza bruta y que nuestros datos permanezcan seguros.

Por otro lado, se debe realizar un estudio con diferentes tamaños de claves para poder cumplir el punto anterior, el cual se debe realizar manteniendo un compromiso entre velocidad de procesamiento criptográfico y seguridad.

Cabe mencionar la complejidad de abordar todos los aspectos en cuanto a seguridad se refiere sobre la Internet de las Cosas y la dificultad de abordar absolutamente todos los problemas que puedan conllevar la ejecución del proyecto que se propone en este Trabajo Fin de Grado en una situación real, debido a que la investigación y realización de este trabajo está limitado en tiempo y cuya autoría ha sido llevada a cabo por una única persona, además de la ayuda del tutor.

1.4 CONTENIDO DE LA MEMORIA

Esta memoria muestra en detalle la realización de un trabajo de diseño de un sistema completo con una funcionalidad específica y de desarrollo del mismo en un entorno simulado mediante el uso de un lenguaje de programación orientado a objetos.

Como parte de un Trabajo de Fin de Grado, se pueden diferenciar mediante la forma de numeración de las páginas en dos grandes partes:

- Índices y Secciones obligatorios en el idioma inglés, incluyendo introducción, resumen extendido y conclusiones en dicho idioma.

- Cuerpo del Trabajo, el cual es la memoria en idioma Castellano.

En cuanto a la parte "Cuerpo del Trabajo", ésta está dividida en varias secciones perfectamente diferenciadas:

- Inicialmente comenzaremos hablando en el Capítulo 1 con estos apartados de Introducción, Motivación, Objetivos y contenido de la memoria.

- En el Capítulo 2 hablaremos sobre el Estado del Arte, explicando el origen del problema, la base sobre la que se apoya y definiendo las tecnologías necesarias para aplicar la solución planteada en este Trabajo de Fin de Grado.

- A continuación, en el Capítulo 3 nos centraremos en el diseño y desarrollo de la solución en sí, el escenario sobre el que se realizarán las simulaciones y pruebas de rendimiento y los problemas que han surgido durante la realización del trabajo.

- Una vez explicados los puntos anteriores, se procede a mostrar los resultados más significativos del proyecto en el Capítulo 4 de una forma limpia y transparente, comentando los mismos y aconsejando/desaconsejando su uso en función de los resultados.

- Tras ello, se incluye un apéndice a la memoria con la inclusión de la planificación, presupuestos y marco económico actual sobre el que se trata el problema.

- Como última sección, se listan todos los enlaces, referencias y bibliografía utilizados durante la realización de este TFG.

CAPÍTULO 2: ESTADO DEL ARTE

2.1 INTRODUCCIÓN

Comenzaremos este capítulo mostrando el origen del problema planteado, tal y como ha sido explicado en el capítulo 1 y mostrando las tecnologías que aplicaremos en el siguiente capítulo, el cual muestra la solución a dicho problema.

Como ya se mencionó, uno de los grandes problemas que tiene la filosofía de la Internet de las Cosas en cuanto a su implementación es la dificultad de las distintas organizaciones por realizar un prototipo estandarizado de cada elemento; no es difícil encontrar dispositivos del hogar, especialmente bombillas LED[1] capaces de conectarse a Internet vía Bluetooth o por Wi-Fi[2] a una conexión de red inalámbrica proporcionada por un punto de acceso del hogar y poder controlarlas remotamente con una aplicación móvil diseñada específicamente para ese dispositivo en concreto y de ese fabricante en exclusiva, aunque con funciones limitadas como encender/apagar la bombilla, cambiar el color de emisión del LED o incluso reproducir música con un altavoz integrado, estilo hilo musical inalámbrico.

Un problema que puede ocurrir en dicho escenario es, como se ha mencionado, que dicha bombilla funciona únicamente con una aplicación de ese fabricante y con funcionalidades limitadas, de forma que esa misma aplicación no es funcional con otro tipo de bombilla de otro fabricante o incluso puede ocurrir que un modelo más nuevo de la misma tenga otro tipo de aplicación también diferente.

Por ello, inicialmente hay que hablar de la necesidad de estandarización de productos para que la Internet of Things verdaderamente despegue, de lo contrario las incompatibilidades o las dependencias de los hogares a una empresa concreta harán de una mala experiencia en este ámbito y no se podría exprimir al máximo lo que nos puede proporcionar la IoT.

Otra preocupación, más importante que el anterior es la seguridad de los datos intercambiados entre dispositivos de IoT y la aplicación que los gestiona es si verdaderamente son seguros nuestros datos durante el camino. Podría darse el caso de que si se gestionan dentro de un ámbito de red local podamos pensar que existe seguridad, pero no siempre podría darse esa condición en caso de existir otros sistemas desconocidos en mi red o existen piratas informáticos que han roto la seguridad, por ejemplo, de la red Wi-Fi; tampoco si existe malware en alguno de los equipos o dispositivos personales conectados a dicha red inalámbrica.

[1] Véase LED como las siglas de “Light Emitting Diode”, Diodo emisor de luz.
[2] Entiéndase como una red inalámbrica segura, i.e. con seguridad WPA2 con apoyo sobre AES.

Esta es la verdadera motivación de la realización de este trabajo, la preocupación por la seguridad y la de nuestros datos, independientemente de si se realiza dentro de una red local, una intranet empresarial o dichos datos pasan a una Cloud; se ha propuesto una solución a ese problema agregando una capa de seguridad extra al cifrado que ya nos pueda proporcionar la red u otras aplicaciones, y ésta es la realización de un escenario virtual que se asemejará a un entorno real de nodos interconectados de forma asíncrona.

En las siguientes subsecciones comentaremos los aspectos y los detalles más importantes de las tecnologías que se han aplicado a la solución final al problema concreto de la seguridad de los datos que en algún momento atraviesen redes o Clouds que no tienen por qué ser confiables.

2.2 LA INTERNET DE LAS COSAS (IoT)

La Internet de las Cosas (o en inglés the Internet of Things), es lo que se define como la interconexión digital de objetos de cualquier tipo, especialmente los dispositivos del hogar, a Internet, de forma que todos puedan interactuar entre ellos de forma autónoma sin necesidad de la intervención humana para comunicarse [6].

Figura 3. Prueba de concepto de la definición de la Internet de las Cosas. ***[7]***

Aunque no seamos conscientes de ello, ya hay dispositivos y objetos que podrían considerarse como futuristas pero poco se alejan de lo que viene a ser el concepto de la Internet of Things. Un avance importante de investigación en este campo lo realiza la empresa norteamericana Google, la cual publica desde hace un tiempo sus pruebas piloto incluso con pasajeros humanos en su prueba de concepto del coche autónomo sin conductor, lo que la empresa denomina como Self-Driving Car Project [8].

Dicho proyecto se basa en la realización automática de acciones en función de los eventos que, dicho vehículo, va capturando con una cámara situada en el techo del mismo; además de ello, el vehículo está conectado con sistemas de la propia empresa de forma que el sistema está en contínuo avance gracias a los datos que va procesando en tiempo real.

Gracias al todos los elementos activos de los que dispone, el vehículo puede ser un sistema automático que conduce sin la necesidad de la intervención de un ser humano; por ello, este vehículo puede considerarse como uno de los adelantos que nos proporciona la Internet of Things.

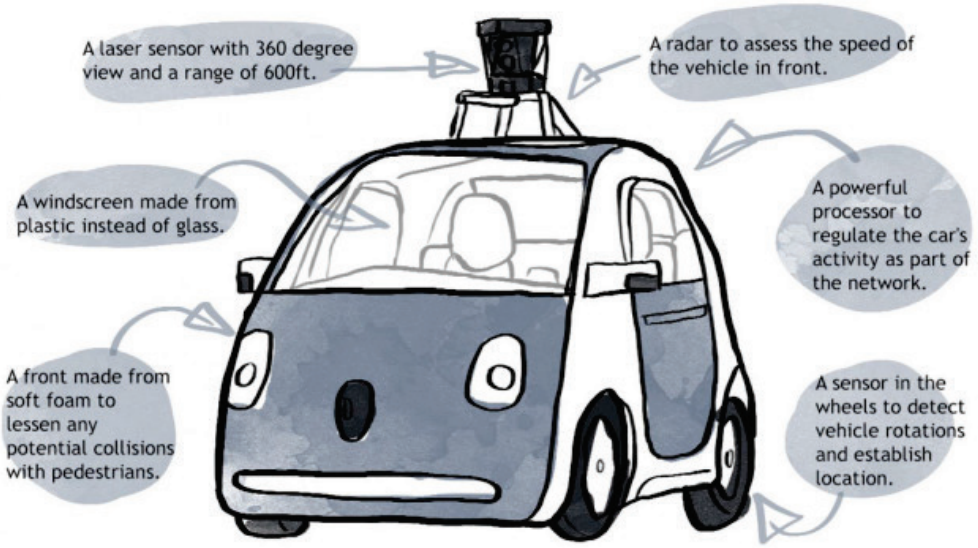


Figura 4. Muestra de los sistemas acoplados al coche autónomo de Google. ***[9]***

Tras todo lo visto anteriormente, nos damos cuenta de la capacidad que ha tenido a lo largo del tiempo el ser humano por ir evolucionando en el diseño de los sistemas informáticos y de las redes de comunicaciones. Desde únicamente disponer del ordenador personal fijo conectado a la red, a múltiples dispositivos conectados a la misma como uno o varios smartphones, tablets, televisores inteligentes SmartTV, consolas, teléfonos VoIP y un largo etcétera.

De hecho en 2009, Kevin Ashton, perteneciente al MIT (Massachusets Institute of Technology) e inventor de uno de los estándares para sensores entre ellos RFID (IDentificación mediante RadioFrecuencia), hizo la siguiente declaración para la revista RFID [10]:

"Los ordenadores actuales, y por tanto, Internet, son prácticamente dependientes de los seres humanos para recabar información. Una mayoría de los casi 50 PB (50 * 1024 GB) de datos disponibles en Internet fueron inicialmente creados por humanos, a base de introducir datos a mano con el teclado. El problema es que las personas tienen tiempo, atención y precisión limitadas."

...

"Si tuviéramos ordenadores que supieran todo lo que tuvieran que saber sobre las "cosas", mediante el uso de datos que ellos mismos pudieran recoger sin nuestra ayuda, nosotros podríamos monitorizar, contar y localizar todo a nuestro alrededor, de esta manera se reducirían increíblemente gastos, pérdidas y costes. Sabríamos cuando reemplazar, reparar o recuperar lo que fuera, así como conocer si su funcionamiento estuviera siendo correcto.

El internet de las cosas tiene el potencial para cambiar el mundo tal y como hizo la revolución digital hace unas décadas. Tal vez incluso hasta más."

Una vez analizada dicha cita, es totalmente cierto que en la actualidad ya existen en el ámbito de la investigación varios procesos mencionados por Kevin Ashton en dicha declaración; procesos como la minería, indexado, modelado y búsqueda de datos para obtener resultados estadísticos, etc...

Pero en la actualidad aún no existe una forma de que las máquinas sean capaz de tomar sus propias decisiones tal y como lo haría un ser humano; siempre existen dudas acerca de qué ocurriría si perdemos el control de dichos sistemas y qué consecuencias puede tener para nosotros que pudiera tomar decisiones propias de forma automática y no lo podamos controlar, algo que la sociedad podría tener cierto miedo de que eso ocurriera en una gran escala.

Mientras tanto, ya que los problemas anteriores son demasiado complejos como para analizarlo de forma tan temprana, comenzamos suponiendo que utilizamos uno o varios sensores de temperatura en un laboratorio. Dichos sensores obtienen la temperatura ambiente con una frecuencia determinada y, supongamos que los envían a un ordenador independiente que almacena dichos datos para después proporcionárselos a un usuario, tanto el valor más reciente como todos los anteriores, mostrando valores estadísticos de la temperatura media a ciertas horas y en cierto día de la semana.

Para llevar a cabo la comunicación, tanto el sensor como la máquina que recibe los datos debe estar conectada a un mismo enrutador, y además de las peticiones automáticas, la máquina recopiladora de muestras (en adelante, Controlador) puede solicitar en cualquier momento nuevas muestras al sensor, independientemente de las que se realizan en modo automático, el sensor también atenderá dichas peticiones exclusivas.

Una de las prácticas más innovadoras actualmente es lo que comúnmente se denomina como **Crowd Sensing** [11]; dicho concepto establece el uso de dispositivos móviles como una fuente de obtención de datos e información de lo que sucede en el entorno, bien para un uso personal como para un uso colectivo.

Una aplicación del Crowd Sensing como uso colectivo de dispositivos móviles es el envío de la velocidad media a la que circulan los usuarios, la zona por la que circulan gracias a su GPS integrado, etc... que implementa por ejemplo la aplicación Infotransit desarrollada por el RACC [12].

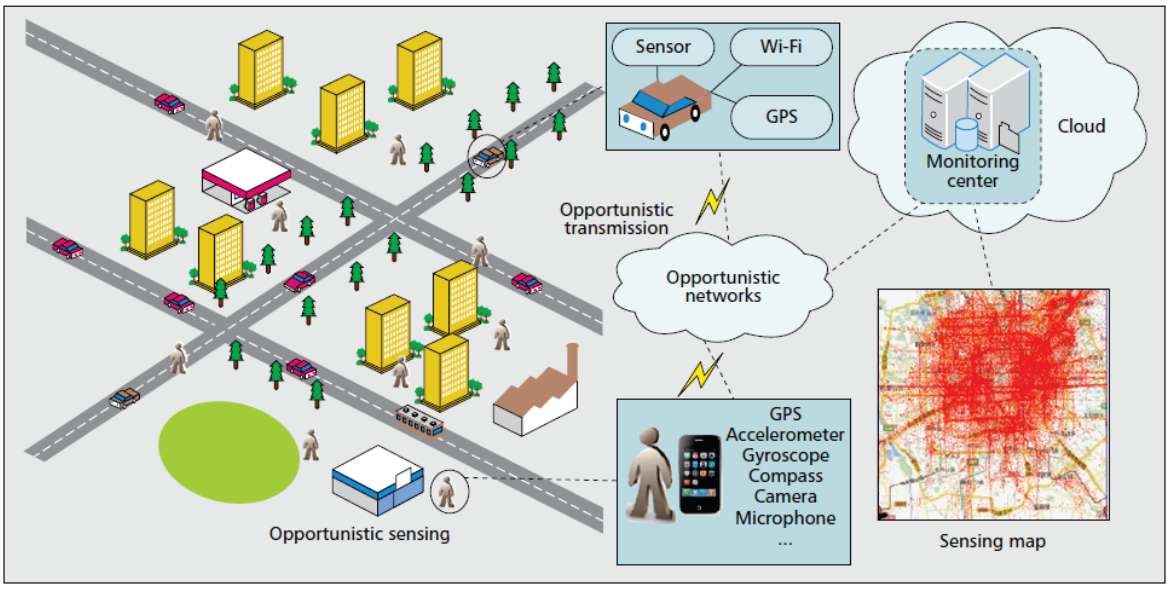


Figura 5. Muestra de varias utilidades de Crowd Sensing en un entorno urbano. ***[11]***

Esta figura muestra un ejemplo de cómo los usuarios pueden contribuir a la recolección de datos de su posicionamiento y otras características del entorno para, por ejemplo, tras procesar los datos, servir a todos los usuarios que deseen utilizar dicho servicio.

2.3 MODELO DE ACTORES

En 1973, Carl Hewitt propuso el concepto de Modelo de Actores [13]; el modelo de programación concurrente o en entornos distribuidos era demasiado complejo de sobrellevar, por ello Carl H. propuso un modelo matemático en el que las primitivas básicas de computación son actores.

En el modelo de actores, cada actor representa a un objeto con un buzón o cola de mensajes a la espera de ser atendidos y un comportamiento específicos.

Por ello, en la práctica, consiste en la utilización de hilos concurrentes denominados en adelante como actores y, gracias a que cada actor está compuesto de un thread o hilo, éste puede estar ejecutándose en paralelo con otros actores de forma simultánea.

Cada actor, por separado, gracias al buzón o cola de mensajes que disponen pueden responder a mensajes que otros actores le envíen, realizar acciones concretas para las que fue diseñado y proporcionar soluciones; además, pueden crear más actores dependientes del mismo a modo jerárquico y decidir cómo responder a otros actores dependiendo de la petición que éste reciba [14].

En este caso, hay que tener en mente la relación de que un actor se va a comportar como un objeto del mundo real, con unas características y un comportamiento determinado en función del objeto al cual esté simulando. Cualquier elemento podría ser emulada y considerada como un actor, desde una simple bombilla como un sistema de alta complejidad, incluso aunque dependa de otras instancias puesto que cada actor puede crear nuevos actores si es necesario, todos ellos siendo componentes o dependencias de un actor principal o “padre”.

2.3.1 COMPORTAMIENTO ENTRE ACTORES:

En el mundo de los actores, este escenario resulta muy familiar, ya que como hemos mencionado anteriormente, simulan a objetos y sistemas reales, esto da lugar a que si cada actor de nuestro entorno ha sido desarrollado con una finalidad concreta y programado adecuadamente, tendrá un comportamiento como dicho sistema lo tendría en la realidad. Ahora vamos a mostrar un ejemplo de dos actores, los cuales simulan un elemento inteligente y un servidor de control respectivamente, siendo éste último al que el usuario propietario del sistema contactará para realizar acciones sobre el entorno. Mostramos la ilustración a continuación:

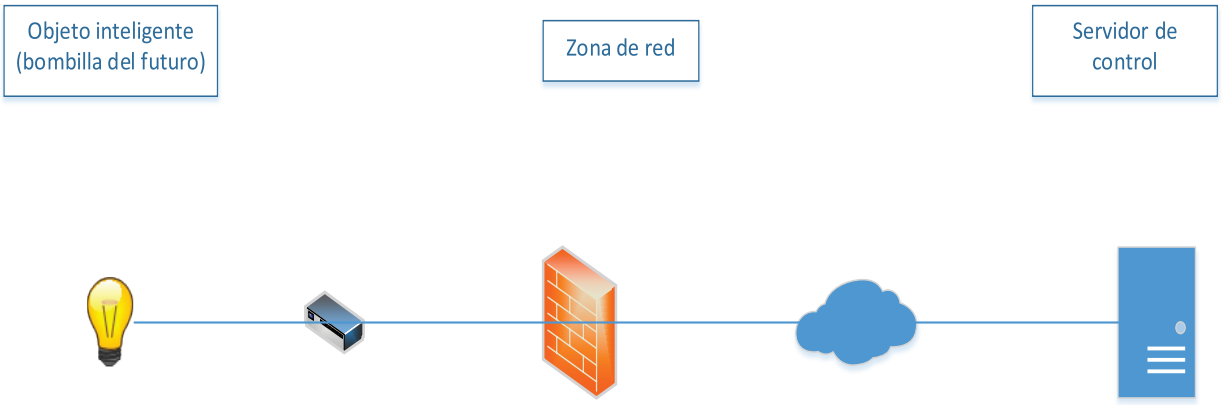


Figura 6. Muestra simplificada de un entorno de modelado de actores.

Obsérvese que ambos extremos serían en este caso actores simulando ambos objetos.

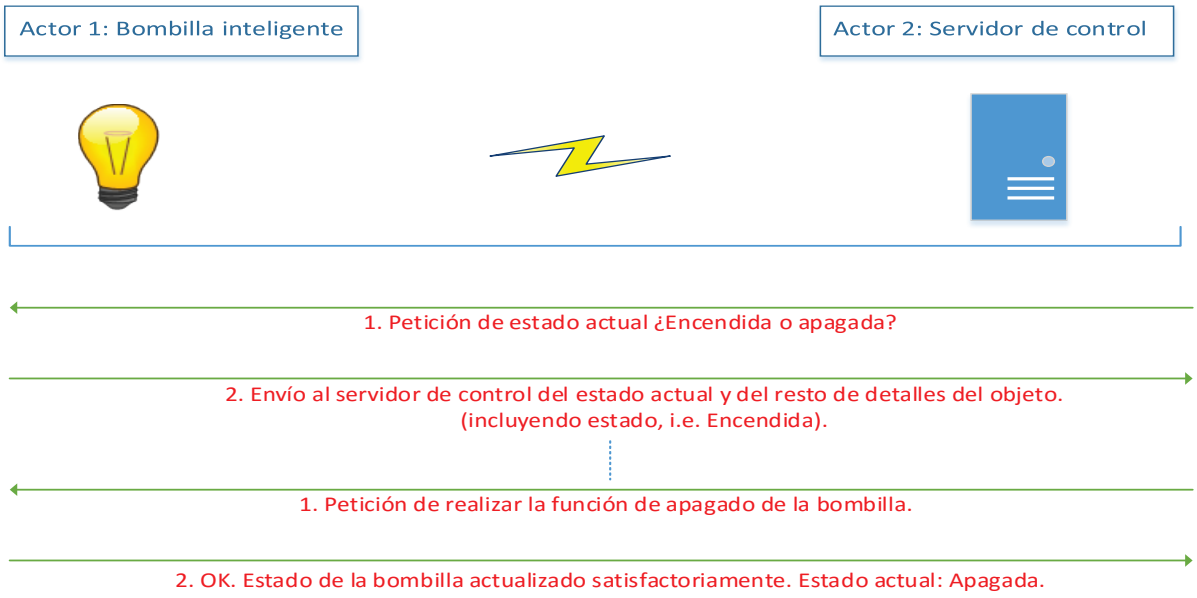


Figura 7. Muestra de ejemplo de comunicación entre actores.

En este ejemplo observamos la semejanza de comportamiento entre actores del entorno especificado (aplicable a otro tipo de objetos, sensores de temperatura, puertas, televisores inteligentes, sistemas de interacción con el usuario, etc...) con las figuras 6 y 7 que explicaban el comportamiento "petición – respuesta" del servicio de correo electrónico.

Este tipo de comportamiento en los actores está basado en lo que es conocido como **Sistema de Actores Reactivos** [15], el cual tenía como misión eliminar la barrera a nivel de diseño que existía hasta hace pocos años, problemática con la gran cantidad de información que se debía procesar y el número de conexiones con peticiones complejas hacían insostenible el mantenimiento de dichos sistemas, los cuales necesitaban tiempo para procesar las peticiones, tardando segundos en atender y responderlas, mantenimiento de la misma por profesionales a horas nocturnas (cuando la demanda era mínima, y se permitía un cierto momento de servicio offline).

En la actualidad, las aplicaciones se pueden crear en cualquier dispositivo, utilizando los recursos de un único elemento (un dispositivo móvil incluso) en aplicaciones pequeñas o de un rack o datacenter entero para una gran aplicación, en la que los usuarios desean que sus peticiones se atiendan inmediatamente y se respondan en cuestión de milisegundos.

Los sistemas reactivos, además de ser de carácter flexible, fácilmente desplegables y escalables, tienen las características siguientes:

- Responsivos: En caso de error, éstos pueden ser reparados de la forma más sencilla posible, incluso si el desarrollador lo desea puede preguntar al usuario qué acciones estaba realizando cuando la aplicación falló, dando cierta confianza a los usuarios de que la aplicación tiene un soporte por detrás de ella. Garantizan un tiempo mínimo de respuesta en la medida de lo posible, incluso podrían caracterizarse que son capaces de ofrecer una garantía en cuanto a calidad de servicio.

- Resilientes: En función de la demanda, estos sistemas son capaces de incrementar o reducir los recursos que necesita para responder adecuadamente a todas y cada una de las peticiones que los usuarios hagan a los mismos; aunque es necesario que dicho sistema esté sobre una base sólida y en un entorno de red en el que no se puedan producir fácilmente pérdidas de paquetes o cuellos de botella.
  Ésto los convierte en unos sistemas con cierta elasticidad, puesto que si dedicamos varias máquinas o servidores en clúster para una aplicación, en función de la demanda usamos más o menos recursos de todas ellas, lo que se traduce en un ahorro de energía y de vida útil de dichos sistemas, porque solo se usan cuando la demanda lo requiere.

- Orientados a Mensajes: éstos establecen una comunicación asíncrona de mensajes, siendo éste el único medio de comunicación entre sistemas; bien se trate de peticiones de usuarios, respuestas, mensajes de depuración, logs, registro de errores, etc... Por ello, estos sistemas disponen de un sistema de colas o bandejas de entrada para atender a dichos mensajes y da como resultado en una comunicación no bloqueante, puesto que tarde o temprano se atenderán los mensajes en la mayor brevedad posible.

Figura 8. Los sistemas reactivos, base sobre la que parte el modelado de actores de Akka. ***[15]***

Obviamente, tal y como hemos comentado en el modelado de actores, otro principio de los sistemas reactivos es la capacidad de que los componentes más pequeños pertenecientes a grandes sistemas dependen de las propiedades reactivas de sus contribuyentes, lo que hace más sencillo el despliegue y la interacción entre los distintos elementos que hacen funcionar un sistema completo.

Esto permite modelar un sistema completo que dependa de varios sistemas más pequeños desarrollados por separado y que pueden establecer contacto entre sí para hacer funcionar un gran sistema con la finalidad de obtener un resultado determinado. Todo ello realizado gracias a la ayuda de esta librería de una forma clara, escalable y con un entorno de desarrollo que facilita la solución de errores en dichos pequeños sistemas por separado mientras realizan la función para la cual fueron diseñados.

2.4 LA LIBRERÍA DE AKKA PARA JAVA

En este apartado del trabajo nos centraremos en la definición del entorno de desarrollo de Java y de las librerías necesarias para el diseño de la solución.

Java [16] es un lenguaje de programación orientado a objetos, con la característica principal de que se puede ejecutar en cualquier sistema que disponga de la máquina virtual de Java instalada en el dispositivo; ésta máquina virtual puede ser instalada en cualquier tipo de ordenador independientemente del sistema operativo utilizado (compatible con Windows, Mac OS, Linux, etc...) e incluso en otros dispositivos específicos con capacidad de ejecutar la misma.

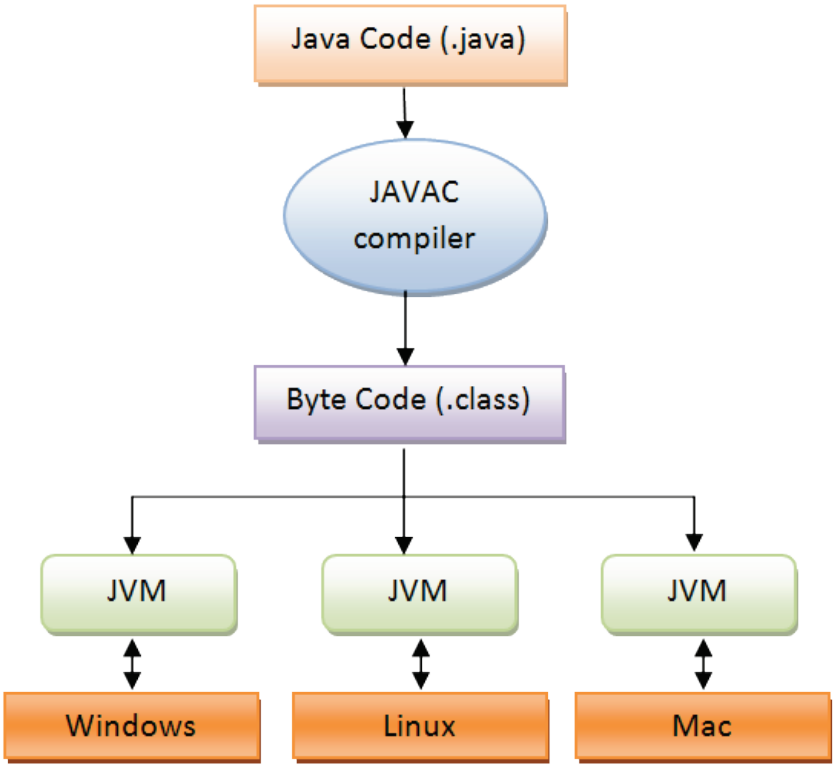


Figura 9. Ejemplo de desarrollo de una clase de java, funcional en varios entornos. ***[16]***

Para el desarrollo de una aplicación en Java es necesaria la descarga de un entorno de desarrollo específico como Eclipse o Netbeans, además del kit de desarrollo del propio lenguaje de programación denominado **JDK (Java Development Kit)** [17] para el sistema en el cual se desean desarrollar aplicaciones.

Una vez instalado todo lo necesario para desarrollar aplicaciones, debemos tener en cuenta si vamos a utilizar funciones ya incluidas por el mismo lenguaje de programación o si además funciones ya compiladas de terceros autores, ya que éstas deben ser importadas en la clase para la cual las vamos a utilizar.

Figura 10. Logotipos de la librería akka y Java. ***[16] [18]***

Akka [18] es una librería Open Source compilada tanto para Java como para Scala (otro lenguaje de programación también orientado a objetos) con la finalidad de aprovechar el potencial del modelado de actores para facilitar el desarrollo de una aplicación compleja que requiera la ejecución de tareas de forma concurrente, tolerante a cambios constantes y escalable, de una forma amigable y basado en el modelo de los sistemas reactivos. Akka pone a disposición a través de su sitio web [18] toda la información necesaria para comenzar a desarrollar desde cero aplicaciones concurrentes sencillas. Simula perfectamente un entorno complejo y de carácter distribuido, aunque sean ejecutados en una única máquina con propósitos de simulación.

En cualquier caso, se puede continuar avanzando en este proyecto, es posible hacerlo independientemente de que existan nuevas versiones de Java o de la librería akka, puesto que se han utilizado durante el mismo funciones sin riesgo de ser descatalogadas o eliminadas de las mismas a corto plazo, o no han sido notificadas de ello por los desarrolladores tanto de Java como de la librería akka.

2.4.1 ARQUITECTURA DE LOS SISTEMAS DE ACTORES EN AKKA:

Antes de crear un actor en sí, es necesario definir uno o varios conjuntos de sistemas de actores. Solo se debería crear una instancia a un sistema de actores [19] por aplicación, puesto que es un objeto de tamaño considerable y creará tantos hilos como considere necesarios para la ejecución de la misma, en función del número de actores creados dentro de dicho sistema.

Los actores en akka forman jerarquías en forma de arbol N-ário (por la razón de que un actor puede crear o no actores hijos, y, de crearlos, podrían ser desde uno hasta los que considere necesarios dentro de la limitación de recursos del sistema.

Dentro de un sistema de actores, la jerarquía base sobre la que se apoyan está ilustrado en la siguiente figura:

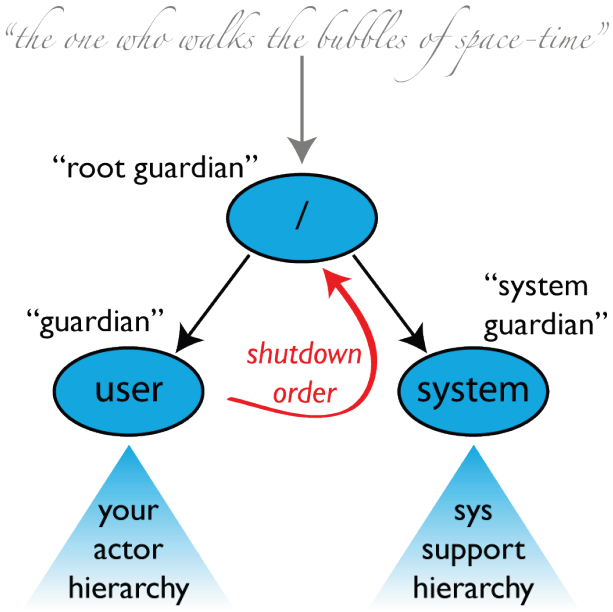


Figura 11. Jerarquía base de un sistema de actores en akka. ***[19]***

Como se puede observar en la ilustración, existen tres actores guardianes principales en cada sistema de actores, los cuales son los siguientes:

- **Guardián raíz:** este actor es conocido en la jerarquía con la ruta "/", éste es el monitor de los otros dos actores guardianes del sistema, responde finalizando un actor hijo de los nodos pertenecientes a la jerarquía "user" en caso de recibir una petición de apagado mediante un mensaje "Terminated", también responde ante cualquier excepción producida en cualquiera de los demás actores.

- **Guardián:** este actor es conocido en la jerarquía con la ruta "/user/", éste es el más utilizado de todos los actores puesto que es el padre de todos los actores generados por el usuario, pasando por el mismo todas las acciones necesarias para que los hijos puedan mantenerse comunicados entre ellos. Cuando éste guardián es finalizado con la acción "Terminated", éste hace finalizar a todos y cada uno de los actores hijos. Si existiera un error en éste guardián, informaría al guardián raíz y éste daría por finalizado todo el sistema de actores por completo.

- **Guardián del sistema:** este actor es conocido en la jerarquía con la ruta "/system/", este actor fue creado para facilitar el orden de apagado de los actores creados por el usuario y para monitorizar a los mismos y al guardián del usuario en caso de errores.

2.4.2 CREACIÓN Y GESTIÓN DE ACTORES POR SU RUTA DE AKKA:

Para crear un actor, una vez definido el sistema al que pertenece, debemos definirle qué instancia debe llevar consigo, siendo necesario que la clase a la que va a contener herede de UntypedActor y disponga de un método específico onReceive() que gestione qué acciones debe realizar el actor al recibir un mensaje.

Al crear un actor se le puede definir un sufijo a su ruta para que sea fácilmente reconocido; estos podrían ser varios ejemplos de una ruta local de actores [20] creados en un sistema llamado "pruebaconactores":

"akka://pruebaconactores/user/actorEjemplo1"
"akka://pruebaconactores/user/sensorTemperatura"
"akka://pruebaconactores/user/controlador"

En caso de que dichos actores estén creados en un entorno virtual de red, las direcciones de akka sufren ligeras modificaciones para que los actores puedan comunicarse entre sí, suponiendo que están en máquinas diferentes dentro de la simulación (observar la diferencia en los números de puerto):

"akka.tcp://pruebaconactores@10.0.1.200:12345/user/ActorEjemplo1"
"akka.tcp://pruebaconactores@10.0.1.201:13456/user/sensorTemperatura"
"akka.tcp://pruebaconactores@10.0.1.199:10001/user/controlador"

Como se puede observar en este caso, nos da a entender que en la simulación existen tres actores simulando tres objetos diferentes en ubicaciones diferentes (direcciones IP y puertos diferentes), lo que da lugar a una solución muy práctica y sencilla de realizar pruebas en un entorno de la Internet of Things con actores simulando varios equipos o sistemas con direcciones IP concretas para cada uno de ellos y por conectividad usando el protocolo TCP con un puerto determinado.

Nótese que al determinar un actor por su dirección de akka remota, se pueden enviar mensajes a dicha dirección incluso aunque se trate de una máquina virtual de java (JVM) remota con akka también instalado y correctamente configurada dicha posible conexión en el fichero application.conf.

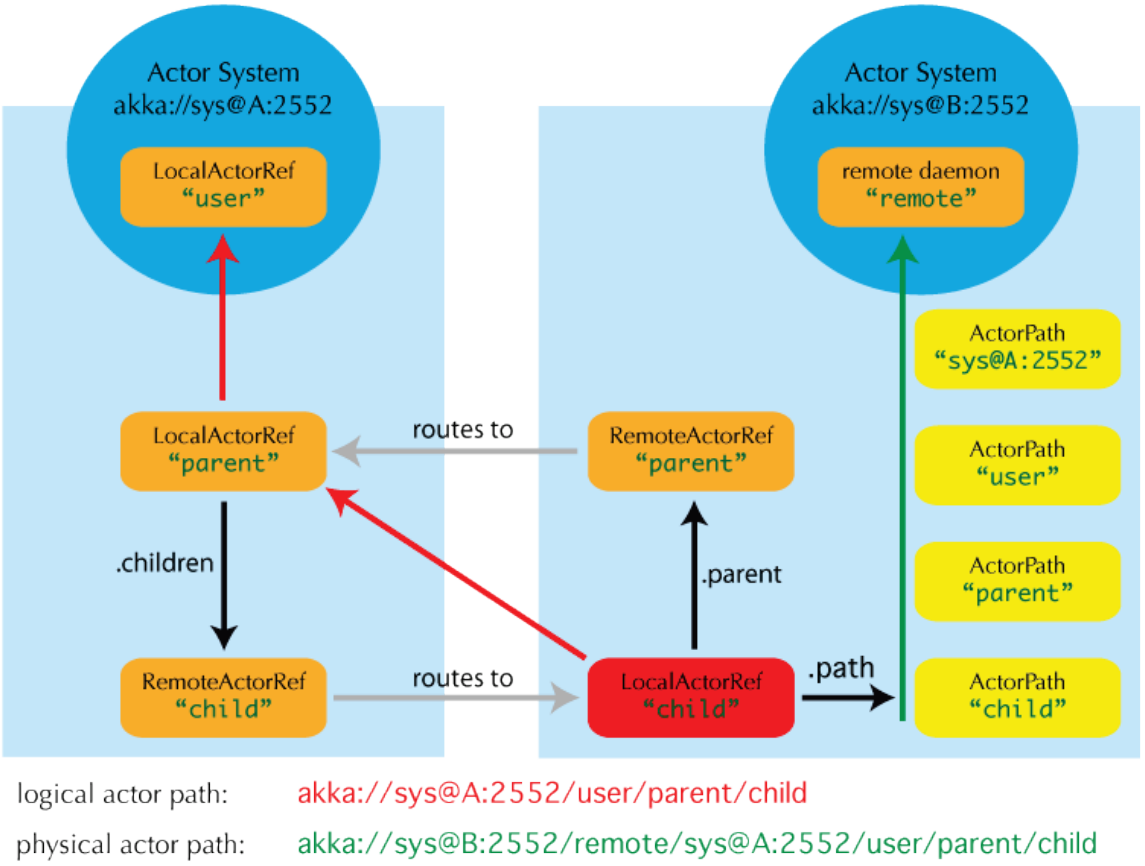


Figura 12. Muestra de la denominación de actores en dos localizaciones simuladas diferentes. ***[21]***

2.4.3 ENVÍO DE MENSAJES LOCALES Y REMOTOS A TRAVÉS DE AKKA:

En caso de tratarse del envío de instancias de objetos, éstos siempre deben implementar la interfaz java.io.Serializable, debido a que al enviar dichos "objetos" en mensajes hacia un destinatario concreto, estos mensajes deben de ser convertidos a un flujo de bytes, enviarlos a la dirección de memoria del destino correcta y en destino reconstruir el objeto.

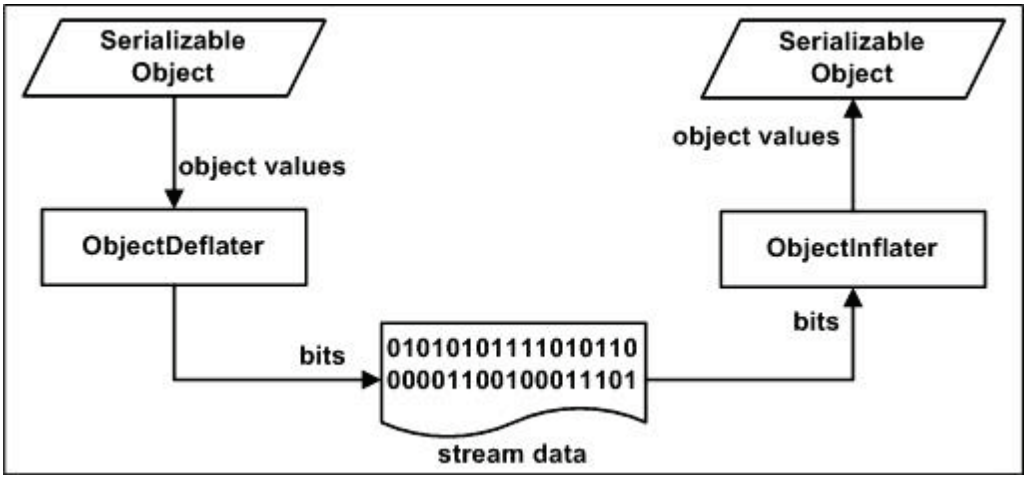


Figura 13. Procedimiento de serialización en origen y deserialización en destino. ***[22]***

Aunque puede verse como una solución fácil, no siempre es trivial y viable, debido a que muchas clases precompiladas por ajenos autores no disponen de la inclusión de la interfaz Serializable o no es posible mediante el serializador por defecto de Java, lo que hace necesaria la inclusión de una nueva librería en el trabajo, ésta librería, entre otras que también serían posibles, es denominada "Kryo".

La librería Kryo [23] contiene un Serializador para Akka capaz de realizar las operaciones de serialización para akka de forma más eficiente que el incorporado en el propio JDK de Java; una de las posibilidades, entre otras, es la de serializar instancias a clases a las que no tenemos acceso al código o no podemos recompilar, por no ser de nuestra autoría, lo cual lo convierte en una herramienta muy potente para el envío de instancias muy grandes o importadas por librerías ya compiladas.

Además, Kryo soporta el cifrado de datos mediante AES y es capaz de serializar prácticamente cualquier tipo de clase de una forma más eficiente y rápida que el serializador tradicional de Java.

Tanto en el actor origen como destino, se debe de crear en el fichero application.conf de akka un conjunto de líneas que especifican un número concreto de serialización para una instancia determinada, de forma que tanto origen como destino son capaces de reconocer qué tipo de instancia es la que se está enviando.

Se entrará en detalle sobre qué parámetros son necesarios modificar y especificar en dicho fichero en el subapartado 3.3 de este Trabajo de Fin de Grado.

2.5 POLÍTICAS DE RECIFRADO MEDIANTE EL USO DE ENTIDADES INTERMEDIAS

2.5.1 PROXY RE-ENCRYPTION

En esta sección del capítulo comenzaremos hablando de la idea del recifrado por un proxy intermedio en una comunicación entre dos extremos y de las variantes criptográficas que existen como posibles soluciones, eligiendo una de ellas como parte de la solución a los problemas de privacidad existentes en la actualidad sobre el futuro de la Internet of Things en este Trabajo de Fin de Grado.

Este escenario de Proxy Re-encryption [24] es aplicable en los casos en los que se desea cierta intimidad sobre el intercambio de claves constantemente los extremos origen y destinatarios de una o varias transmisiones de datos considerados como sensibles o privados, incluso si se desconfía sobre la seguridad de los datos dentro de una misma red local.

Por esta causa, se agrega un tercer elemento que actúa de intermediario y es considerado como un proxy de recifrado, el cual no puede ni siquiera descifrar los mensajes que pasan a través de él mismo, puesto que no dispone de ninguna de las claves privadas necesarias para descifrar el mensaje en sí; tan solo dispone de una clave de recifrado entre dichos extremos que puede, dependiendo del algoritmo utilizado, ser unidireccional o bidireccional.

En el caso de que se trate de una clave de recifrado unidireccional, es necesario que el proxy disponga de dos claves de recifrado por cada conexión entre extremos si estos quieren comunicarse mutuamente, mientras que en el caso de que se trate de una clave de recifrado bidireccional, serían necesarias únicamente una clave por cada comunicación entre extremos, independientemente del sentido en el que estén transmitiéndose los mensajes en ese instante de tiempo.

Este tipo de algoritmos proporcionan una capa de seguridad adicional a la que los propios protocolos de red o aplicación ya proporcionen a dicha comunicación, asegurando a su implementor una mayor garantía de confidencialidad en la transmisión de datos sensibles a través de canales de transmisión sobre los cuales no se tiene el control de ellos directamente, por pertenecer a un dueño particular diferente o incluso una empresa de la competencia que pueda estar interesada en obtener nuestros datos confidenciales.

Existen varios algoritmos de seguridad que utilizan este modelo, encontrando información principalmente sobre dos de ellos, BBS y AFGH, entre otros:

- **BBS (Blaze, Bleumer & Strauss, 1998)** [25]: es un algoritmo basado en la forma criptosistemática de ElGamal e introduce el concepto de Clave de Recifrado. Aplicar dicha clave de recifrado a un texto cifrado hace que éste pase de poder ser descifrado con una clave privada del cifrante a poder serlo con la clave privada del destinatario, todo ello posible sin conocer el texto plano ni ninguna de las dos claves privadas de los extremos.
  Este algoritmo dota al proxy la capacidad de calcular la clave de recifrado para ambos sentidos, con solo conocer la de uno de ellos puesto que es el inverso multiplicativo de la misma en cuerpo q, aunque en la práctica éste almacena dos claves de recifrado, una para cada sentido.

- **AFGH (Ateniese, Fu, Green & Hohenberger, 2005)** [26]: al igual que BBS, también está basado en la forma criptosistemática de ElGamal e introduce una nueva seguridad en el concepto de la Clave de Recifrado, la imposibilidad del proxy de calcular la clave del sentido opuesto de comunicación entre dichos extremos, ya que no cumple la propiedad de que sea el inverso de la que ya dispone. Otra diferencia respecto a BBS es que AFGH está basado en las propiedades de los Cuerpos de Galois o Bilinear Maps.

Tras analizar las ventajas e inconvenientes de ambos, se ha optado por elegir AFGH por tener una mayor complejidad y dotar del mínimo de "poder" al proxy de valerse por sí mismo para calcular partes del sistema de la capa de seguridad que se desea aplicar a el concepto de la Internet of Things.

2.5.2 EL ALGORITMO AFGH, CARACTERÍSTICAS Y TEORÍA MATEMÁTICA

La principal características de AFGH [26] es la posibilidad de intercambiar información entre extremos sin la necesidad de establecer entre ellos un secreto compartido. Tan solo es necesario intercambiarse entre ellos mediante una comunicación directa las claves públicas generadas por ambos, para así calcular la clave de recifrado necesaria para cada sentido, en cada uno de ellos y enviársela al proxy para su posterior procesado de mensajes.

Por ello comenzamos a explicar cómo funciona este algoritmo a nivel de aritmética modular y una de las librerías de las que depende el mismo, la librería JPBC (Java Pairing-Based Cryptography Library) [27].

Utilizaremos por simplicidad de aquí en adelante la librería NICS (Network Information and Computer Security) [28], que engloba el uso de AFGH de una forma sencilla utilizando los pasos siguientes:

❖ ***Paso 1: Generación de la curva elíptica y sus parámetros característicos.***

En este trabajo se ha decidido implementar cifrado mediante curvas elípticas y bilinear maps, los cuales existen de varios tipos según la librería JPBC: *A*, *A1*, *D*, *E*, *F* y *G*. Para este caso, se han utilizado los de tipo *A*.

Sea una curva elíptica cuya ecuación general es de la forma:

$y^2 = x^3 + ax + b$ con *'a'* y *'b'* constantes y debe cumplirse que $4a^3 + 27b^2 \neq 0$. **(2.01)**

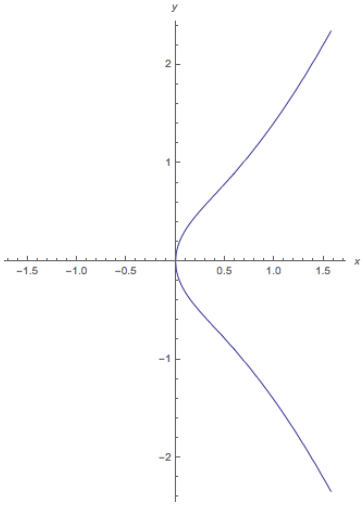


Figura 14. Dibujo exp. 2.02.

La curva elíptica del tipo A es una solución concreta de la ecuación anterior con $a = 1\ y\ b = 0$ de forma que se cumple que: $4 * 1^3 + 27 * 0^2 = 4 \neq 0$.

Dicha ecuación de curva elíptica es:

$$y^2 = x^3 + x \quad \in \mathbb{Z}_q \ para\ un\ primo\ q = 3\ mod\ 4 \qquad \textbf{(2.02)}$$

Para ello se usan los Cuerpos Finitos o Cuerpos de Galois [26] sobre un primo *'q'*, en adelante denominados 'Bilinear Maps'. Para este caso, son los grupos de puntos $\mathbb{G}_1$ y $\mathbb{G}_2$ que pertenecen a $E(\mathbb{F}_q)$.

Se cumple la siguiente propiedad sobre $E(\mathbb{F}_q)$ y $E(\mathbb{F}_q{}^2)$:

$$E(\mathbb{F}_q) = q + 1 \qquad \text{y} \qquad E(\mathbb{F}_q{}^2) = (q+1)^2 \qquad \textbf{(2.03)}$$

Tanto el número primo *'q'* como *'r'*, son de tamaño variable en función del número de bits que sean asignados en el momento de su generación, más adelante se comprueba la fortaleza del algoritmo probando tamaños en incremento de cada uno de estos números y, por supuesto, de las claves generadas con los mismos.

El primo *'q'* es un número que cumple, independientemente de su tamaño en longitud y bits, que es el mayor valor posible en cuerpo 4. $q = 3\ mod\ 4$

El número *'r'*, en este tipo de parámetros de curva tipo *A*, es elegible por el usuario, que en este caso, cumple que debe ser un Número Primo de Solinas.

Un número Primo de Solinas [29] es aquel cuyo valor cumple la siguiente expresión:

$$r = 2^a \pm 2^b \pm 1 \quad / \quad a, b \in \mathbb{Z}_q \quad y \quad 0 < b < a \qquad \textbf{(2.04)}$$

Que en adelante, tanto en este informe como en el código fuente del trabajo, será denotada como:

$$r = 2^{exp2} + sign1 * 2^{exp1} + sign0 * 1 \qquad \textbf{(2.05)}$$

Siendo $exp2 = a$, $exp1 = b$, $sign1 = \pm 1$ y $sign0 = \pm 1$.

Una vez obtenidos ambos números, *'q'* y *'r'*, gracias a la librería "AFGHGlobalParameters" podemos calcular los mismos de una manera rápida, sencilla y automática los valores necesarios para la generación de las claves publicas, privadas, método de cifrado/descifrado y las claves de recifrado.

El tiempo necesario para la generación de los parámetros varía en función del número de bits utilizados para los primos *'q'* y *'r'*, unos números pequeños darán un resultado de encontrar unos primos *'q'* y *'r'* más rápidamente que si se utilizan primos más grandes, por lo que es de gran utilidad guardar dichos números para regenerar solo los parámetros y no tener que recalcularlos si no fuera necesario (a menos que se hubiese comprometido la seguridad), pues disminuye drásticamente el tiempo de generación de los parámetros y el tiempo útil de funcionamiento normal del sistema para atender peticiones de generación de claves.

- ***Paso 2: Generación de pares de claves pública y privada.***

En este trabajo, para cada actor, independientemente de si es un elemento de la IoT, un proxy o un receptor se generan su correspondiente par de claves. Dentro de ese par se encuentra la clave privada, en adelante denotada como SK_α y una clave pública, denotada como PK_α, cuyos valores son:

$SK_\alpha = \alpha$, siendo α un número entero aleatorio dentro del conjunto $\mathbb{Z}_q$ **(2.06)**
$PK_\alpha = g^{SK_\alpha} = g^\alpha$ **(2.07)**

Esto es común a multitud de algoritmos de criptografía asimétrica, siendo 'El Gamal' el algoritmo sobre el que se basa AFGH, con la modificación de que este último profundiza aún más aprovechando las propiedades que nos proporciona la

exponenciación y su facilidad para sumar y restar exponentes con la misma base, en este caso, el generador *'g'* perteneciente a $\mathbb{G}_1$ y *e(g,g)* ó *'Z'* perteneciente a $\mathbb{G}_2$.

Se cumple una propiedad importante de los Cuerpos de Galois y que veremos más adelante en la aplicación de la clave de recifrado a un mensaje, y es que la exponenciación de los elementos generadores en ambos cuerpos da como resultado 'Z^x' siendo x el producto de los exponentes, véase:

$$\begin{aligned} e(g,g) &= \mathrm{Z} \\ e(g^a,g) &= \mathrm{Z}^a \\ e(g^a,g^b) &= \mathrm{Z}^{ab} \end{aligned} \qquad \textbf{(2.08)}$$

Lo cual facilita las operaciones entre distintos cuerpos finitos.

❖ ***Paso 3: Generación de la clave de recifrado usando AFGH.***

Una vez generadas las claves pública y privada en un nodo origen o destino, para llevar a cabo la comunicación y aprovechar los beneficios que nos proporciona AFGH es necesario calcular la clave de recifrado única que servirá al proxy como herramienta necesaria para recifrar criptogramas cuyo origen y destino sean los que intervinieron en la generación de dicha clave, y cuyo sentido de comunicación también está claramente definido. A causa de ello, es trivial que es necesario calcular dos claves de recifrado entre extremos para que la comunicación sea en ambos sentidos, dejando al proxy la responsabilidad de recifrar con la clave correcta al sentido en el cual se está enviando el criptograma en ese momento.

Por ello, como algoritmo de cifrado unidireccional que es, dejamos claro que siempre se cumple que $RK_{\alpha_1 \to \alpha_2} \neq RK_{\alpha_2 \to \alpha_1}$ **(2.09)**, puesto que las probabilidades de que las claves públicas y privadas de ambos extremos sean las mismas son ínfimas.

Una vez entendido este concepto, procedemos a explicar la forma en la que uno de los extremos inicia el proceso de creación de la clave de recifrado:

- Inicialmente, ambos extremos disponen de sus claves pública y privada correspondiente. Denotaremos estos extremos como α_1 y α_2, respectivamente.
- En un determinado instante de tiempo, α_1 inicia una conexión directa con α_2 solicitándole su clave pública, momento en el cual le contesta a α_1 con dicha clave PK_{α_2}.
- Una vez recibida PK_{α_2} por α_1, éste genera la clave de recifrado con sentido $\alpha_1 \to \alpha_2$, $RK_{\alpha_1 \to \alpha_2}$ gracias a dicha clave pública remota PK_{α_2} y a su privada SK_{α_1}.

- El cálculo de dicha clave de recifrado se calcula como:

$$RK_{\alpha_1 \to \alpha_2} = PK_{\alpha_2}{}^{1/SK_{\alpha_1}} = g^{\frac{SK_{\alpha_2}}{SK_{\alpha_1}}} = g^{\frac{\alpha_2}{\alpha_1}} \quad \textbf{(2.10)}$$

- Ésta clave, una vez calculada, es enviada al servidor proxy encargado de recifrar los posteriores criptogramas cuyo sentido sea $\alpha_1 \to \alpha_2$.

Nótese que este proceso ha sido mostrado suponiendo el sentido de comunicación entre los sistemas de $\alpha_1 \to \alpha_2$, es trivial que para el caso opuesto, el proxy deberá recibir desde α_2 una clave de recifrado como la siguiente, en el caso de tratarse de una nueva comunicación entre $\alpha_2 \to \alpha_1$:

$$RK_{\alpha_2 \to \alpha_1} = PK_{\alpha_1}{}^{1/SK_{\alpha_2}} = g^{\frac{SK_{\alpha_1}}{SK_{\alpha_2}}} = g^{\frac{\alpha_1}{\alpha_2}} \qquad \textbf{(2.11)}$$

Tras este apartado, quedaría explicado el funcionamiento de la creación de claves de recifrado.

Ahora, comencemos a analizar el tratamiento de los mensajes desde que salen de un origen hasta que llegan a su destino, supongamos el escenario anterior en el cual se intenta establecer una conexión entre $\alpha_1 \to \alpha_2$, ambos extremos disponen de la siguiente información hasta este momento:

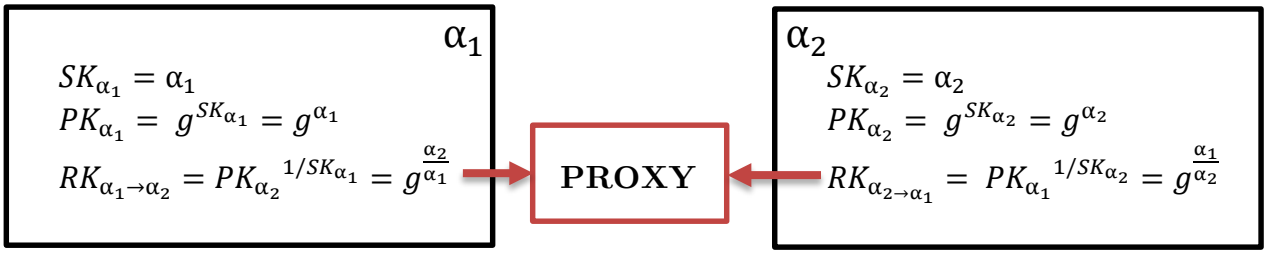


Figura 15. Diagrama de muestra de claves que se comparten con el Proxy.

Supongamos que en un momento determinado, α_1, desea enviar un mensaje M con sentido α_2.

❖ **Pasos de Cifrado y Descifrado del algoritmo AFGH:**

- Cifrado del mensaje M en α_1 , realizando las operaciones siguientes, denotaremos el criptograma C con subíndices en función de si ha atravesado el proxy o aún no lo ha hecho, luego, el criptograma C_1 resulta como:

$$C_{1_{\alpha_1 \to \alpha_2}} = [A_1, B_1] = [g^{\alpha_1 * r}, M * Z^r] = \left[PK_{\alpha_1}{}^r, M * Z^r\right] \qquad (2.12)$$

- Una vez cifrado, este criptograma viaja hasta el proxy, el cual tras detectar el origen y el destinatario, aplica la clave de recifrado correspondiente:

$$C_{2_{\alpha_1 \to \alpha_2}} = [A_2, B_2] = \left[e\left(A_1, RK_{\alpha_1 \to \alpha_2}\right), B_1\right] = \left[e\left(g^{\alpha_1 * r}, g^{\frac{\alpha_2}{\alpha_1}}\right), M * Z^r\right] = [Z^{r * \alpha_2}, M * Z^r] \quad (2.13)$$

- Una vez recibido en el destinatario α_2 el criptograma $C_{2_{\alpha_1 \to \alpha_2}}$, éste procesa a descifrarlo de la siguiente manera:

En α_2: $$\frac{B_2}{A_2{}^{1/SK_{\alpha_2}}} = \frac{B_1}{A_2{}^{1/SK_{\alpha_2}}} = \frac{M * Z^r}{Z^{r * \alpha_2/\alpha_2}} = M \quad (2.14)$$

Lo que demuestra la eficacia de las propiedades del algoritmo AFGH, ya que el mensaje M ha sido recuperado en el destino incluso pasando por dos capas de cifrado totalmente diferentes, y sin necesidad de transmitir el mensaje con ninguna de las claves públicas del receptor. Cuando α_1 cifró el mensaje, lo hizo con su propia clave pública, no con la de α_2. En ese supuesto, α_1 puede fácilmente descifrar su propio criptograma utilizando la expresión:

En α_1: $$\frac{B_1}{e(A_1, g^{\frac{1}{SK_{\alpha_1}}})} = \frac{M * Z^r}{e(g^{\alpha_1 * r}, g^{\frac{1}{SK_{\alpha_1}}})} = \frac{M * Z^r}{Z^{\alpha_1 * r/\alpha_1}} = M \quad (2.15)$$

En cuanto a la forma en la que se cifran los mensajes, en general todas las tramas deben ir en segmentos con una cierta longitud de bytes, lo cual hace necesario para toda transmisión es la transformación de objetos a algo que se pueda cifrar, como lo es en este trabajo la decisión de transformar todos los objetos susceptibles de ser enviados como cadenas o Strings; esto permite que un objeto sea transformado a cadena con una función toString(), le sean a la misma aplicadas transformaciones de cifrado, recifrado y descifrado, y, en última instancia, la reconstrucción a objeto mediante un constructor específicamente diseñado para ello. Esto garantiza la correcta recuperación del objeto si realmente se era el destinatario de dicho mensaje, de lo contrario no será posible descifrarlo y dará como resultado una cadena con caracteres ilegibles y por ello la imposibilidad de reconstruir el objeto.

Nota: una de las ventajas de AFGH, al contrario que otros algoritmos basados en criptografía asimétrica como 'RSA' o la base de 'El-Gamal', es la posibilidad de recuperar el mensaje cifrado por el nodo cifrante, incluso habiendo re-cifrado el mensaje el intermediario, gracias a que la componente cifrante $B_2 = B_1$, cada extremo tan solo tiene que utilizar la expresión de descifrado final correcta y podrá descifrar el mensaje; esto es posible debido a que el mensaje es cifrado con su propia

pública y no con la ajena, lo cual permite al cifrante recuperar el mensaje realizando la operación inversa al cifrado utilizando su propia clave privada.

Otra de las características de este algoritmo es que no se establece en ningún momento una clave precompartida, lo que comúnmente se denomina en criptografía el "secreto compartido". La comunicación entre dos nodos extremos puede realizarse sin ninguna negociación previa de un secreto compartido, ni siquiera la clave de recifrado que recibe el proxy de un sentido es transmitida entre extremos, lo que mejora la fortaleza del sistema ante atacantes y suplantaciones de identidad una vez iniciada la comunicación.

El Proxy nunca será capaz de descifrar ningún mensaje, independientemente del origen/destino de los mismos y del instante de tiempo en el que ocurra, porque no dispone de ninguna de las claves privadas necesarias para descifrarlos. Dicha entidad ni siquiera dispone de las claves públicas, que serían la única pista para obtener la privada correspondiente, lo que descarta incluso intentos de ataques de fuerza bruta o la motivación de comprometer la seguridad del proxy para intentar hacerse con el control y espionaje de las comunicaciones. Los objetos entran cifrados, se procesan criptogramas y vuelven a salir cifrados, en ningún momento se procesa con texto plano.

En el caso de la comunicación entre α_1 y α_2 por comodidad en esta explicación, supongamos que el proxy ha sido comprometido y un atacante desea interceptar los mensajes recibidos por el mismo. El proxy solo dispone de las direcciones origen y destino de los nodos extremos y de la clave de recifrado correspondiente $RK_{\alpha_1 \to \alpha_2} = g^{\frac{\alpha_2}{\alpha_1}}$ **(2.10)**, aunque el proxy sí que puede conocer $C_{2_{\alpha_1 \to \alpha_2}}$ **(2.13)**, que es el criptograma que enviará al destino α_2 necesario para descifrar el mensaje, no dispone de la clave privada α_2, no es en absoluto factible conseguir α_2 a partir de la clave de recifrado $g^{\frac{\alpha_2}{\alpha_1}}$ **(2.10)**, lo que hace que este algoritmo sea resistente a colisiones y redundancias, suponiendo que conocieramos la privada de α_1 también sería bastante complicado despejar α_2 en cuerpo q, especialmente en claves de tamaños considerables de bits.

Una limitación que nos da AFGH es el dobre cifrado nativo del algoritmo en una única transmisión. Supongamos que añadimos un tercer nodo extremo α_3 y que el proxy ya ha podido anteriormente tener contacto con el mismo, teniendo disponibilidad de todas las claves de recifrado posibles desde y hacia α_3.

Un escenario en el que α_1 desea enviar hacia α_3 un mensaje cifrado con su propia clave pública pero con una marca de que desea recifrarlo como si fuera en el

siguiente orden: $\alpha_1 \rightarrow \alpha_2 \rightarrow \alpha_3$ pero realizando dichas operaciones criptográficas en el proxy.

Pues bien según la limitación de AFGH, **no es posible** realizar la siguiente sucesión de operaciones:

$$C_{1_{\alpha_1 \rightarrow \alpha_2 \rightarrow \alpha_3}} \rightarrow RK_{\alpha_1 \rightarrow \alpha_2} \rightarrow C_{2_{\alpha_1 \rightarrow \alpha_2 \rightarrow \alpha_3}} \rightarrow RK_{\alpha_2 \rightarrow \alpha_3} \rightarrow C_{3_{\alpha_1 \rightarrow \alpha_2 \rightarrow \alpha_3}} \quad \textbf{(2.16)}$$

Esto es debido a la propiedad de los mapas bilineales o cuerpos finitos de Galois.

En el criptograma $C_{1_{\alpha_1 \rightarrow \alpha_2}}$ **(2.12)**, tenemos dos elementos: ambos pertenecen a un cuerpo finito perteneciente a $\mathbb{G}_1$, mientras que tras realizar la operación de recifrado en el proxy, al generar $C_{2_{\alpha_1 \rightarrow \alpha_2}}$ **(2.13)** el segundo elemento permanece intacto, lo que continúa siendo parte del cuerpo finito $\mathbb{G}_1$, mientras que el primer elemento debido a que ha sido transformado con la función 'e', pertenece a un segundo cuerpo finito $\mathbb{G}_2$, el cual el destino también conoce dicho cuerpo y podrá operar sobre él.

Esto quiere decir que si se realizase el doble recifrado, al reoperar se generaría un segundo elemento del nuevo criptograma basado en otro cuerpo finito diferente $\mathbb{G}_3$, que el destino no sería capaz de interpretar, lo que genera una gran ventaja y es que demuestra autenticidad del mensaje debido a que no permite la delegación automática de los mensajes a través del proxy; a menos que dichos mensajes lleguen a su destino y vuelvan a ser cifrados con origen y destinos correctos, el nuevo destino no será capaz de descifrarlo.

CAPÍTULO 3: DISEÑO Y DESARROLLO

3.1 INTRODUCCIÓN

Una vez ya explicadas todas las cualidades que va a necesitar la solución planteada, para la creación de la capa adicional de seguridad se utilizarán las funciones que nos proporcionan las librerías JPBC y NiCS, para la correcta aplicación del algoritmo AFGH en la comunicación con nuestros actores que simularán nuestro entorno de la Internet of Things.

En cuanto al soporte del modelo de actores, como ya hemos mencionado, utilizaremos Akka y con la ayuda del serializador Kryo podremos realizar una comunicación correcta entre los mismos, la que nos permitirá serializar cualquier clase que queramos enviar entre cualesquiera de los actores.

Para comenzar a diseñar y desarrollar dicha solución, se ha decidido utilizar como soporte una máquina virtual formateada con Windows 10 Pro, 3 GB de RAM, disco de estado sólido SSD, y dedicación de dos núcleos lógicos de un procesador i5-2400S a 2.49 GHz.

3.1.1 USO DE AKKA PARA MODELAR INTERNET OF THINGS (IoT):

Una vez explicado en Capítulo 2: Estado del arte las tecnologías a las que se hará referencia en este capítul, se observa que el uso Akka junto con Kryo hacen en conjunto una gran base sobre la que se puede asentar prácticamente cualquier simulación sobre prototipos en los que se desee una comunicación o albergar un sistema de control en un futuro para la IoT, puesto que cualquier elemento puede ser albergado en un actor por muy complejo que sea y cumple con la necesidad de los mismos de estar conectados a la vez de forma asíncrona, de esta manera podemos tener una comunicación con la absoluta certeza de que todos los mensajes serán recibidos por el destinatario de cada uno de ellos, siempre y cuando estemos dentro del entorno de pruebas.

Más adelante hablaremos de la integración de akka con varias librerías criptográficas que serán utilizadas para el diseño del sistema completo, y su comportamiento entre la comunicación de los actores si se les añade una capa extra de seguridad.

3.2 PLANTEAMIENTO Y DISEÑO

Inicialmente, comenzamos a explicar el entorno que se desea simular. A continuación se muestra una ilustración acerca de un entorno general de red susceptible a ser modificado según sea necesario, modificando las propiedades de cada bloque o agregando nuevos en un futuro.

En la ilustración mostrada en la Figura 16 localizada en la página siguiente, se pueden observar cuatro tipos diferentes de actores, entre ellos podemos observar los siguientes:

- Actores que actúan de clientes: son los que simulan ser un elemento de la 'Internet of Things', como pueden ser una bombilla, un televisor inteligente, un Sensor de Temperatura enviando información constantemente sobre la información ambiente o una puerta, entre otros elementos.

- Actor servidor generador de parámetros de curva elíptica de tipo A y de pares de claves: este actor tiene la finalidad de calcular los números primos y el cálculo de los parámetros de los Bilinear Maps G_1, G_2 y los parámetros 'g' y 'Z', necesarios para el cómputo de claves públicas, privadas y posteriormente en los nodos origen y destino, la de recifrado unidireccional.

- Actor servidor intermediario de recifrado o Proxy Re-encrypter: este actor se encarga de recibir todos los mensajes ya cifrado, realizar la operación de recifrado y proceder a entregar a su destinatario el nuevo criptograma; además de recibir nuevas claves de recifrado y almacenarlas en su memoria. Este actor nunca podrá ver el contenido en plano de los mensajes, pues no dispone de ninguna clave pública ni privada de los extremos.

- Actor destino o servidor: este actor simula ser el actor que interactúa con la plataforma online / servidor externo el cual bien almacena en una base de datos todos los detalles de los elementos o interactúa mediante un interfaz Rest a las peticiones de los usuarios autorizados.

Nota: IPSEC o TLS son sistemas de seguridad extremo a extremo que también presentarían utilidad de cifrado para casos similares de forma síncrona, pero no son de aplicación para un entorno de este tipo. En un entorno de Crowd Sensing pueden llegar a estar enviando muestras o paquetes millones de elementos, siendo el coste de comunicar todos ellos de forma síncrona y en tiempo real enorme; esa es la razón por la que se aplica esta solución, ya que es aplicable a comunicaciones en las cuales los elementos se comunican de forma asíncrona.

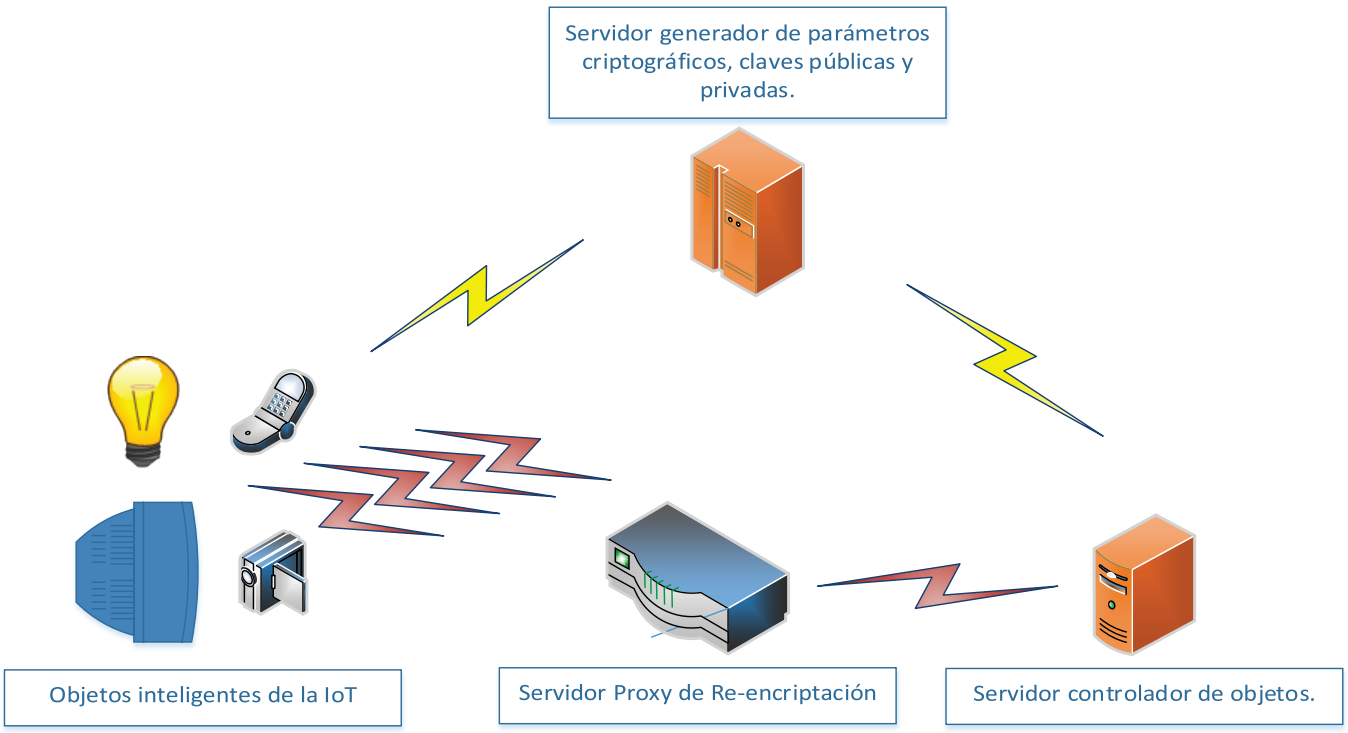


Figura 16. Diseño del escenario final de pruebas de la solución de este TFG.

Por ello, podemos observar que hay dos tipos diferentes de "conectividad"; las marcadas en amarillo son conexiones que ya se pueden considerar a priori seguras, puesto que el servidor generador de parámetros criptográficos además de las claves públicas y privadas que son enviadas a cada nodo es considerado como externo y de una fuente de certificación considerada como segura de antemano.

Por otro lado, observamos en la Figura 16 las conexiones entre objetos, proxy y servidor controlador como un tipo diferente a las anteriores, debido a que consideramos que dicho canal posiblemente no sea tan seguro como puede decir que lo es el proveedor de servicios de internet o las redes intermedias que los paquetes de datos atraviesen. Por ello, dicha capa de seguridad adicional solo afecta a los paquetes enviados entre:

Objetos – Servidor Proxy Re-encrypter – Servidor controlador

Lo que debe estar siempre en línea y no debe estar offline bajo ninguna circunstancia es el Servidor generador de Parámetros y Claves SK y PK (por ello se podrían tener disponibles varios servidores generadores de claves simultáneamente y en proveedores diferentes), para garantizar la inclusión de nuevos elementos al escenario de pruebas, de lo contrario, éstos no serían capaces de recibir nuevos pares de claves.

A continuación, explicaremos qué información y de qué tipo se intercambia en cada elemento para comunicarse con el resto, tanto cifrada como en texto plano.

- **Paso 1: Inicio del Generador de Parámetros y Claves *SK* y *PK*.**

Figura 17. Servidor generador de parámetros y claves SK y PK.

Este actor debe estar arrancado en primer lugar, comienza a generar los números primos *'q'* y 'r'. Dichos números serán de un tamaño concreto de bits en función del número de bits que se les quiera dedicar a los mismos, según la librería NiCS acerca del algoritmo AFGH, se recomiendan utilizar números con el siguiente número de bits, pudiendo elegir el que más interese según las pruebas de seguridad que se deseen realizar:

Número de bits de 'q'	Número de bits de 'r'
512 bits	**160 bits**
1536 bits	**256 bits**
7680 bits	**512 bits**

Tabla 1. Diferentes tipos de fortalezas para AFGH.

De forma que en función del nivel de seguridad y de la dureza que deseemos que emplee el algoritmo AFGH, eligiremos unos valores más altos o más bajos de dichos números. En el Capítulo 4: Resultados se muestran las diferencias en tiempo necesario de generación de dichos números y de los parámetros de la curva elíptica, además de los números generadores *'g'* y *'Z'*. Aunque, en principio, podemos adelantar que, a mayor número de bits de los números anteriores, mayor tiempo de computación será necesario por dicho actor para crearlos.

Nota: no supone ningún problema que entren nuevas peticiones de claves mientras se estén generando las de otro elemento, puesto que entran en espera y serán atendidas por orden de llegada, sin despreciar ninguna de ellas.

- **Paso 2: Petición de Claves por parte del Controlador de Objetos y otros elementos nuevos en cualquier instante de tiempo.**

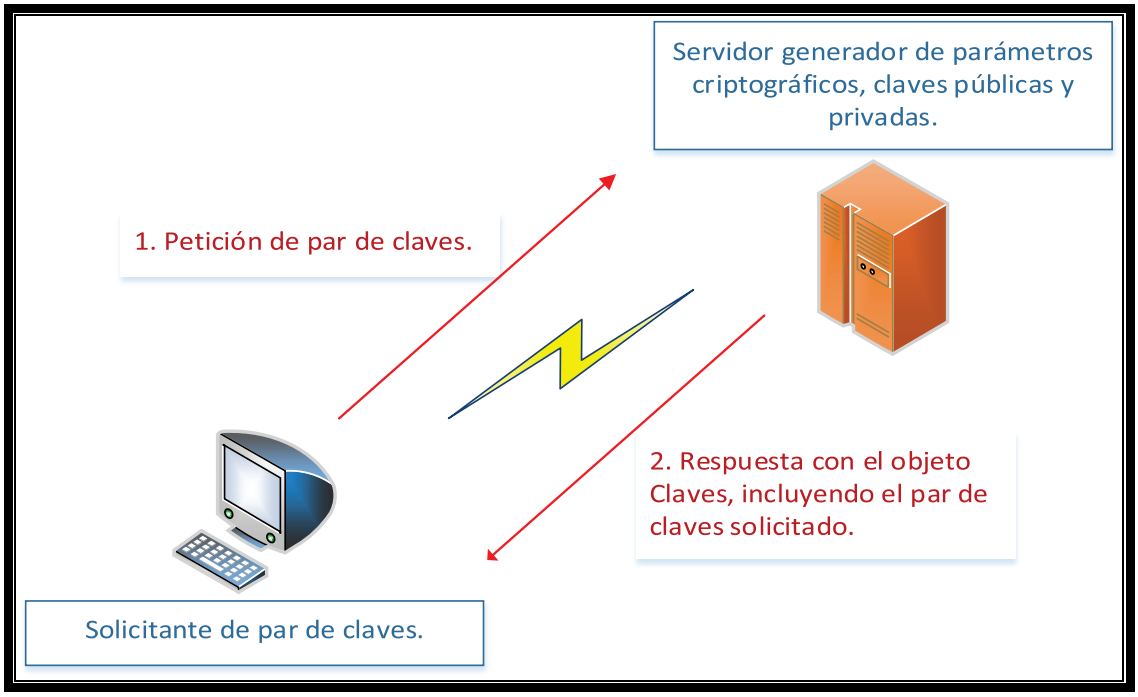


Figura 18. Muestra de solicitud/respuesta de par de claves SK y PK.

Al iniciar un nuevo actor, bien sea el que simula el Controlador de Objetos o cualquier otro nuevo elemento que se incorpore al entorno de simulación, éste enviará una petición de generación de claves al Actor Generador con la siguiente estructura de trama (de aquí en adelante, se omiten las cabeceras TCP y Ethernet, tan solo se muestra el contenido del apartado de Payload de las mismas):

1	Dirección akka remota del actor desde el cual se piden las claves i.e. "akka.tcp://sistema@10.0.1.199:10001/user/control"

Tabla 2. Trama de petición de pares de claves.

Al cual, una vez procesada por el Actor Generador, en adelante denominado como "Keygen", envía una contestación al solicitante del par de claves con la siguiente estructura de trama:

Instancia de la clase "Claves", cuyo contenido es el siguiente: - Clave Privada: *RK* - Clave Pública: *PK* - Dirección Akka del actor propietario de dichas claves. - Ídem al anterior pero en formato String. - Cadena con información de regeneración de parámetros.	Dirección akka remota del Keygen

Tabla 3. Trama de respuesta de pares de claves.

Una vez recibidas dichas claves, los actores quedan a la espera de recibir órdenes por parte del controlador, aunque deben inicialmente de compartir su clave pública de forma directa entre elemento y servidor controlador, para generar en cada extremo la clave de recifrado correspondiente en cada sentido, ya que como hemos hablado en apartados anteriores, las claves de recifrado basadas en el algoritmo AFGH son unidireccionales y, por ello, es necesario que el proxy reciba las claves de reencriptación creadas para ambos sentidos para la correcta comunicación entre los dos extremos para los que fueron creadas.

- **Paso 3: Generación de las Claves de Re-cifrado.**

Para explicar este paso, previamente se muestra una ilustración de apoyo a los conceptos que se explican a continuación:

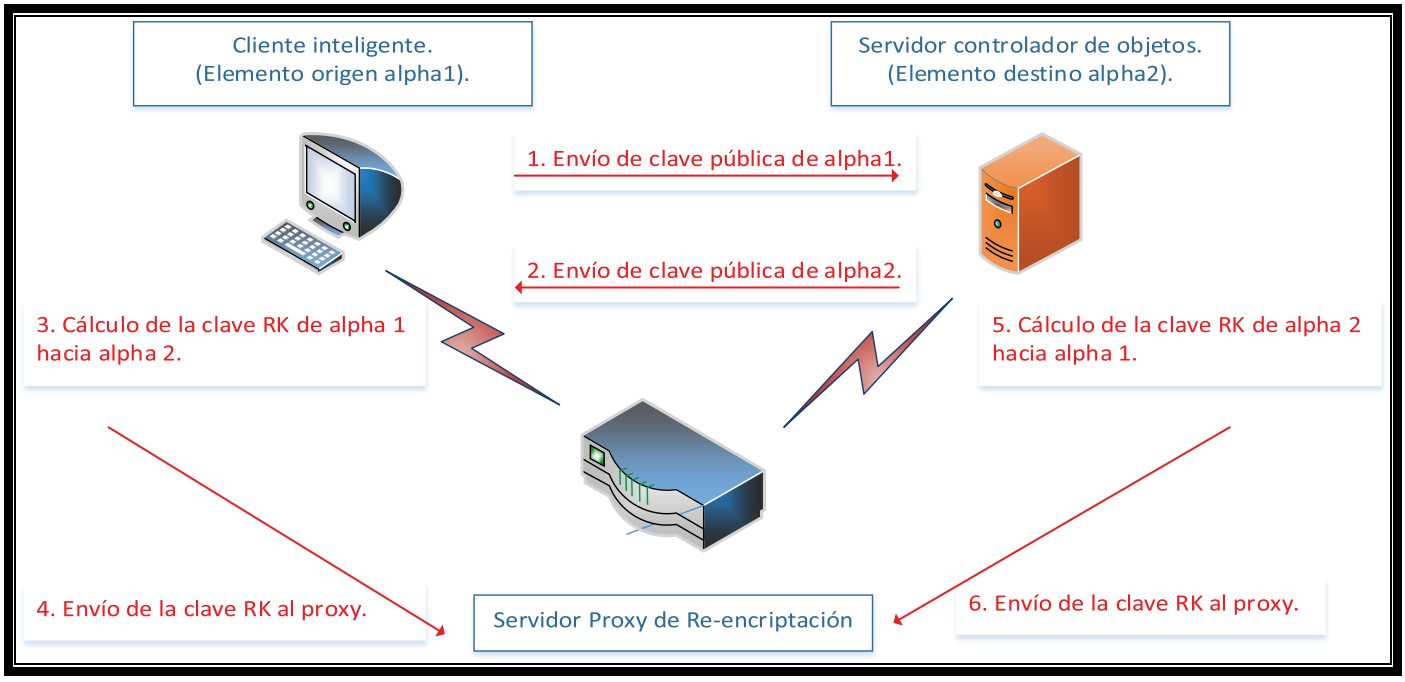


Figura 19. Muestra de intercambio de claves públicas y generación de RK's.

Para realizar la acción de generar claves de recifrado, según hemos visto en la teoría del algoritmo AFGH, es necesario disponer de la clave pública del otro extremo para poder calcularla, tal y como hemos visto en la expresión **(2.10)**, la cual recordemos que era la siguiente:

$$RK_{\alpha_1 \to \alpha_2} = PK_{\alpha_2}{}^{1/SK_{\alpha_1}} = g^{\frac{SK_{\alpha_2}}{SK_{\alpha_1}}} = g^{\frac{\alpha_2}{\alpha_1}} \qquad \textbf{(2.10)}$$

Puesto que suponemos que ya disponemos del par de claves correspondiente en el actor origen, inicialmente debemos hacer una petición al actor con el cual queremos comunicarnos con el siguiente contenido de trama:

Instancia de la clase "Peticion", cuyo contenido es el siguiente: - Dirección akka origen. - Dirección akka destino. - Tipo de petición: 0 (de clave pública del otro extremo) Estos campos muestran al actor destino el sentido sobre la cual será calculada.	Dirección akka remota del solicitante

Tabla 4. Trama de petición de clave pública remota.

A dicha trama de petición, por decisiones de diseño, se ha elegido el valor '0' como entero que determina que dicha petición es una solicitud de la clave pública del actor que la recibe.

Por ello, el actor remoto contesta al que hizo la petición con la siguiente trama, incluyendo la clave pública del mismo, además, éste pide la clave pública con una petición idéntica a la que recibió solo que cambiando de sentido los parámetros "Dirección akka origen" y "Dirección akka destino".

La contestación que envía y recibe un actor, el cual desea iniciar una comunicación con un extremo es la siguiente, con la clave pública del otro extremo:

Instancia de la clase "ClavePublicaAjena", cuyo contenido es: - Clave pública del otro extremo: *PK*. - Dirección akka del propietario de dicha clave *PK*.	Dirección akka del otro extremo

Tabla 5. Trama de respuesta de clave pública remota.

Ahora que ambos extremos tienen en su disposición de la clave pública del contrario, podemos crear la clave de recifrado correspondiente a cada sentido desde el actor correspondiente para satisfacer la expresión **(2.10)**.

Por ello, una vez calculada, cada extremo envía la clave de recifrado calculada por el mismo al Proxy Intermediario de Reencriptación (en adelante, el Proxy), de forma que el mismo dispondrá del par de claves necesario para establecer la comunicación:

Instancia de la clase "ClaveReencriptacion", cuyo contenido es: - Clave de Recifrado calculada: *RK*. - Dirección akka del origen de uso de dicha clave. - Dirección akka del destino de uso de dicha clave.	Dirección akka del creador de la clave de Recifrado

Tabla 6. Trama de transmisión de clave de recifrado nueva.

- **Paso 4: Comunicación entre los elementos y el servidor controlador de objetos, usando el proxy como intermediario.**

Para explicar este paso, previamente se muestran dos ilustraciones de apoyo:

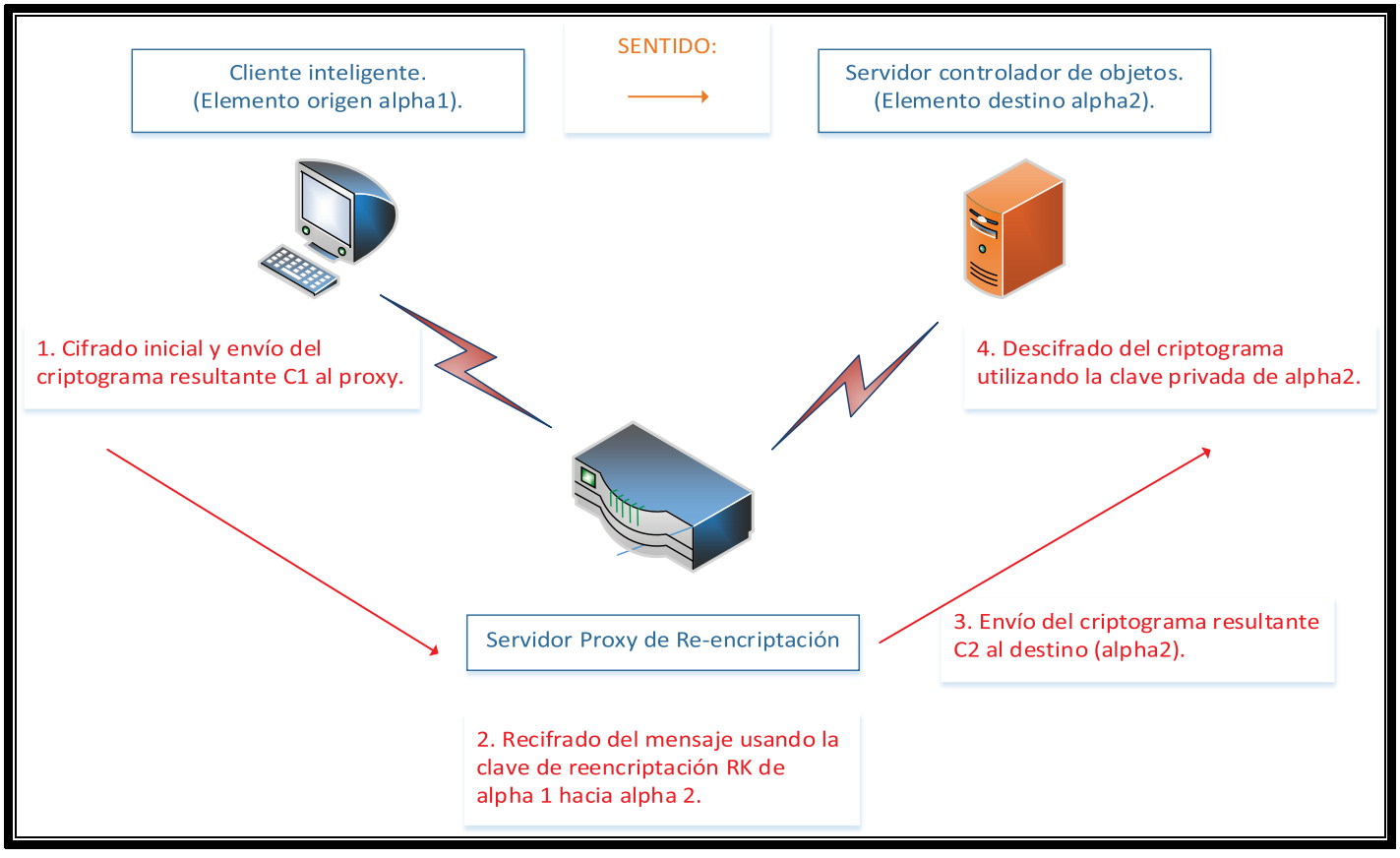


Figura 20. Ejemplo de comunicación asíncrona en sentido 1->2.

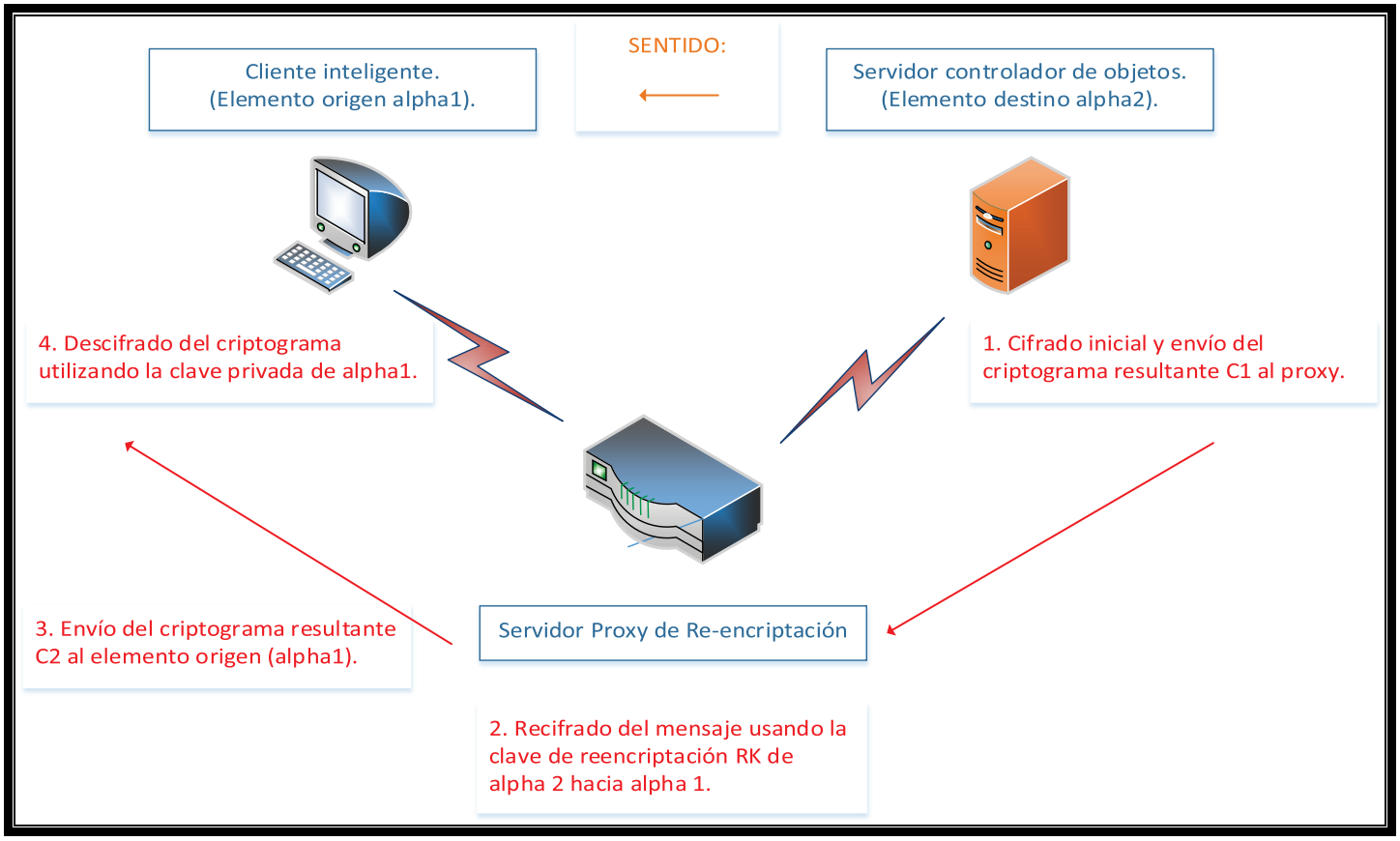


Figura 21. Ejemplo de comunicación asíncrona en sentido 2->1.

Una vez que todos los elementos disponen de sus pares de claves y el Proxy dispone de las claves de recifrado necesarias para establecer la comunicación entre dichos elementos y el servidor de control, es posible enviar mensajes con cualquier tipo de contenido sobre el canal, de una forma más segura gracias a la capa de seguridad proporcionada por el algoritmo AFGH.

Pero el uso de esta seguridad y para evitar problemas de serialización, se ha decidido utilizar como parte del contenido de cada trama, una cadena con un formato concreto que se mostrará a continuación, la cual se convertirá en un criptograma y dicho resultado es el que será enviado por el canal.

Para ello, el contenido de la cadena en texto plano es el siguiente:

Clase	Acción (entero)	ID del elemento	Nombre del elemento	Payload de acciones (sub-cadena)

Tabla 7. Contenido de una cadena de acciones en texto plano.

En función del tipo de objeto al cual simula el actor que genera dicha trama, éste agregará en el payload de acciones el resto de la cadena necesaria en función de

la acción que éste va a realizar. Por ejemplo, en un sensor de temperatura, si éste desea actualizar la temperatura y enviarla al servidor, enviará una cadena con el contenido:

$$M = \text{“SensorTemperatura|1|14|Sensor de Ejemplo|36”}$$

Parseando dicha cadena observaríamos que la trama envía una petición de tipo 1, enviada desde el Sensor de Temperatura con ID 14 y cuya temperatura actual recogida por el mismo es de 36 grados centígrados. De esta forma, el Servidor de control parseará dicho mensaje una vez descifrado y tras ello realizará las acciones necesarias.

Tras este breve ejemplo, mostramos la trama resultante (que incluye el criptograma ya cifrado) que será enviada hacia el proxy (paso 1 de la Figura 20):

Instancia de la clase “CriptogramaAFGH”, cuyo contenido es: - Elementos del Tuple que componen el criptograma. - Dirección akka “origen” del mensaje. - Dirección akka “destino” del mensaje.	Dirección akka del creador del mensaje

Tabla 8. Trama de envío de mensaje cifrado hacia el proxy.

Supongamos que el sensor de temperatura “origen” genera la petición de actualización de la temperatura detectada por dicho sensor que será recibida más adelante por el “destino” controlador de objetos. Sucedería lo siguiente:

Paso 4.1: El actor que simula un sensor de temperatura genera la cadena resultante de normalización como se propuso en el diseño de la comunicación.

Paso 4.2: Éste mismo actor, aplica el algoritmo AFGH cifrando dicha cadena, aplicando la ecuación mostrada en la expresión **(2.12)** que se muestra a continuación:

$$C_{1_{\alpha_1 \to \alpha_2}} = [A_1, B_1] = [g^{\alpha_1 * r}, M * Z^r] = \left[PK_{\alpha_1}{}^r, M * Z^r\right]$$

Siendo α_1 el actor que representa al sensor de temperatura y α_2 el actor que representa al servidor controlador de objetos.

Nota: Observe que M es la cadena generada anteriormente en el paso 4.1.
Este escenario es el aplicado en el paso 1 de la figura 20.

Paso 4.3: El proxy recibe el criptograma $C_{1_{\alpha_1 \to \alpha_2}}$ (la trama que incluye una instancia de la clase "CriptogramaAFGH") y procede a realizar el proceso de recifrado; éste lee los campos sin cifrar "Origen" y "Destino" de la trama recibida para elegir la clave de recifrado correcta que ya debería tener almacenada en su base de datos.

Una vez seleccionada correctamente, el proxy se dispone a tramsformar el criptograma recibido $C_{1_{\alpha_1 \to \alpha_2}}$ en un nuevo criptograma $C_{2_{\alpha_1 \to \alpha_2}}$, cumpliendo la transformación mostrada en la expresión **(2.13)** (paso 2 de la figura 20):

$$C_{2_{\alpha_1 \to \alpha_2}} = [A_2, B_2] = \left[e\left(A_1, RK_{\alpha_1 \to \alpha_2}\right), B_1\right] = \left[e\left(g^{\alpha_1 * r}, g^{\frac{\alpha_2}{\alpha_1}}\right), M * Z^r\right] = [Z^{r*\alpha_2}, M * Z^r]$$

Por ello, es trivial deducir el formato de la trama que enviará el proxy hacia el destinatario contenido en el campo "destino", ya que es exactamente igual a la trama que el proxy ha recibido, solo que con el criptograma ya actualizado y el campo de la dirección akka actualizado con la referencia al proxy (paso 3 de la figura 20):

Instancia de la clase "CriptogramaAFGH", cuyo contenido es: - Elementos del Tuple que componen el criptograma. - Dirección akka "origen" del mensaje. - Dirección akka "destino" del mensaje.	Dirección akka del Proxy

Tabla 9. Trama de envío de mensaje cifrado desde el proxy hacia el destino.

Una vez recibida dicha trama por el destinatario original, éste procede a deshacer el cifrado, utilizando para ello su clave privada tal y como se puede ver en la expresión **(2.14)**:

$$\frac{B_2}{{A_2}^{1/SK_{\alpha_2}}} = \frac{B_1}{{A_2}^{1/SK_{\alpha_2}}} = \frac{M * Z^r}{Z^{r*\alpha_2/\alpha_2}} = M$$

De esta forma, el destinatario es capaz de recuperar el mensaje propuesto en el último ejemplo (paso 4 de la figura 20):

$$\boldsymbol{M = \text{"SensorTemperatura|1|14|Sensor de Ejemplo|36"}}$$

En la Figura 21 se puede observar el proceso de comunicación en el sentido

opuesto, es trivial que el procedimiento es totalmente inverso teniendo en cuenta que la clave de recifrado a utilizar por el proxy es $RK_{\alpha_2 \to \alpha_1}$, la cual fue proporcionada por α_2 al proxy como se muestra en el 6 de la Figura 19.

En el siguiente apartado se comentarán los distintos aspectos de implementación de los formatos de trama propuestos y de los problemas que han surgido durante la implementación de toda la solución.

3.3 DESARROLLO DE LA SOLUCIÓN

Tras haber dejado claro el planteamiento del problema y el diseño de la solución se procede a mostrar el desarrollo de la solución con las herramientas que nos proporcionan tanto las librerías definidas anteriormente como los entornos de desarrollo o IDE de la máquina virtual de Java.

3.3.1 REQUISITOS PREVIOS AL DESARROLLO

Inicialmente es necesario disponer de un equipo con un sistema operativo compatible con el JDK de Java. Por lo general, al tratarse de un lenguaje de programación compatible con múltiples plataformas no debe suponer ningún problema, indepentientemente de que se desee desarrollar en Windows, cualquier distribución de Linux o en Mac OS X.

En este trabajo se ha decidido utilizar la versión 1.8.0_40 tanto de la máquina virtual de Java como del Kit de herramientas de desarrollo (JDK) en la versión de Windows 10 Pro; como entorno de desarrollo se ha utilizado Netbeans en su versión 8.0.2, compatible con Java SE y EE.

En cuanto a las librerías para Java, es necesario importar los siguientes paquetes de librerías en el proyecto:

- Akka. Versión 2.3.9.
- NiCS. Versión 1.0.
- JPBC.Versión 1.3.0.
- Kryo. Versión 3.0.0.

Por ello, los enlaces de descarga de todas las utilidades y librerías mencionadas son los siguientes (todos los enlaces son de libre descarga):

Máquina virtual de Java y Netbeans:
https://www.java.com/es/download/
https://netbeans.org/downloads/

Kit de Herramientas de Desarrollo (JDK) de Java:
http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html?ssSourceSiteId=otnes

Resto de librerías en los enlaces de cada sitio web: [18], [28], [27] y [23].

3.3.2 CONFIGURACIONES PREVIAS ANTES DE COMENZAR A DESARROLLAR EN AKKA PARA JAVA:

Al descargar e importar la librería de Akka en el proyecto, es necesario editar el fichero **application.conf** localizado dentro de la carpeta config, la cual a su vez se encuentra en la carpeta raiz de la librería akka; este fichero contiene información sensible acerca de las funciones de depuración que se pueden aplicar en la nueva aplicación, informar a akka de si se va a realizar una simulación con actores en un entorno puramente local o un entorno de red con múltiples dispositivos; también es posible especificar el puerto asociado a cada tipo de actor en concreto (bien uno definido por el usuario o indefinido con el valor 0), entre otras configuraciones que serán más adelante entradas en detalle.

En caso de utilizar un entorno de red, es necesario identificar también en dicho fichero las clases cuyas instancias son susceptibles a ser enviadas por un protocolo de red, a esto se le conoce como Serialización de objetos y es necesario que ambos actores dispongan de acceso a dicha instancia, de lo contrario, se podría perder el contenido del mensaje cuyo contenido es una instancia a ese objeto no declarado en el actor destino.

Una vez importado el paquete, se puede proceder al uso de la aplicación y de la ejecución de los sistemas en el orden deseado, pudiendo editar las variables de debug según la prueba que se desee realizar y su exportación si así se desea en ficheros .csv, dichos ficheros pueden importarse en Matlab o Excel y generar tablas de datos con gráficas de tendencia, cálculos estadísticos de tiempos medios, etc...

3.3.3 ESTRUCTURA DE CLASES DEL PROYECTO:

En cuanto a la estructura de clases del proyecto en Netbeans, en adelante 'ProxyReencryptionTFG', se ha decidido ir separando en diferentes paquetes con clases Java de forma que cada paquete contiene el código fuente necesario para ejecutar un actor que simula un objeto.Dichos paquetes son los siguientes:

- **alex.destinationblock:** este paquete contiene el código fuente que simulará ser el controlador de objetos; dicho controlador será el que mantenga la comunicación con el resto de actores que simulen objetos finales (bien sensores de temperatura, bombillas, dispositivos móviles realizando crowd sensing...).

- **alex.keygen:** tal y como se mencionó en capítulos anteriores, el Keygen es el actor responsable de la generación de parámetros de curva elíptica, *'g'*, *'Z'*, y de los pares de claves que otros actores puedan pedir al mismo.

- **alex.objetos:** este paquete contiene los diferentes tipos de objetos (nota: en este caso se trata de instancias a clases, no confundir estos objetos con elementos del mundo real) que se utilizan en las diferentes comunicaciones entre actores; pueden ser peticiones, criptogramas, pares de claves públicas, privadas o de recifrado...

- **alex.originblock:** este paquete contiene el código fuente que simulará ser uno o varios elementos de la internet of things, los cuales interactuarán tanto con el Keygen, con el controlador y con el Proxy.

- **alex.proxy:** este paquete contiene los ficheros relacionados con el sistema recifrante o Proxy, que se encarga de realizar las operaciones de recifrado de los mensajes intercambiados entre extremos origen y destino.

- **alex.tipos:** en este se encuentran distintas clases que son susceptibles de ser consideradas como objetos de la vida real, tanto actuales como del futuro, siendo éstos totalmente modificables y con la posibilidad de añadir nuevos si fuera necesario en un futuro.

- **pruebas:** en este paquete se localizan ficheros puntuales con fines de depuración o de test para que el desarrollador pueda realizar todas las pruebas que considere necesarias sin influir en el resto de ficheros del proyecto.

- Los paquetes **nics.crypto** y **nics.crypto.proxy.afgh** son clases parte de la librería sin precompilar NiCS, utilizando dichos ficheros como si de librerías precompiladas en .jar se tratasen.

Tras definir todos y cada uno de los paquetes, debemos tener en cuenta que el arranque de cada tipo de actor viene definido por la ejecución de una única clase con método main disponible en cada uno de ellos. Esto nos hace posible ver en consolas separadas la salida de cada actor de manera independiente, lo que contribuye a una mejor experiencia en las tareas de depuración del código durante la ejecución de cada uno de ellos.

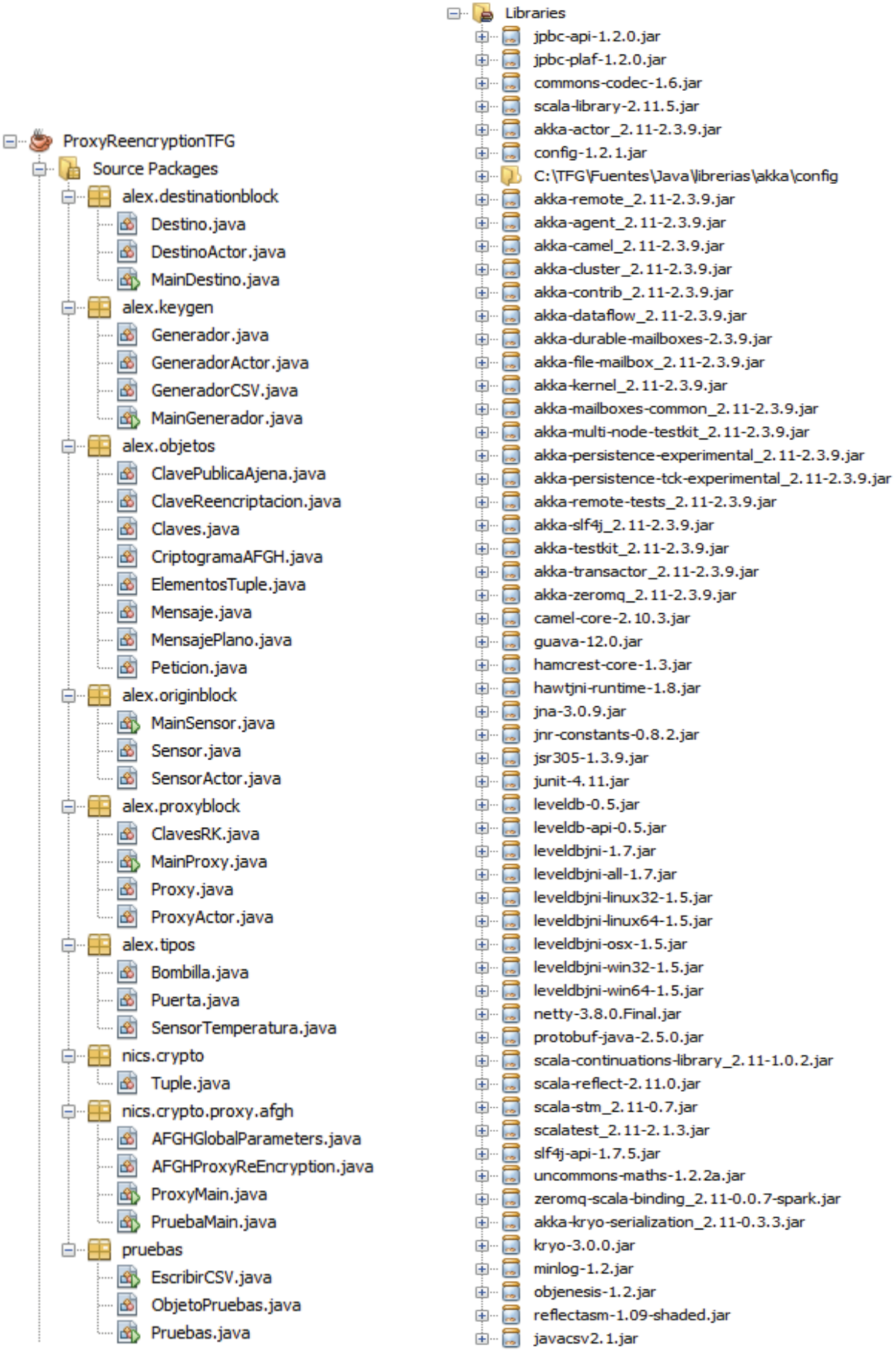


Figura 22. Estructura del proyecto en Netbeans.

3.3.3.1 Bloque Keygen (o Generador):

Este bloque, como ya ha sido comentado, es un actor que simula un sistema de generación de parámetros y de pares de claves.

Este sistema, tal y como ha sido diseñado, es un sistema con espera puesto que atiende todas sus peticiones de forma ordenada según el modelo de actores.

Antes de ser creado el actor, es necesario determinar el tamaño de los números *'q'* y *'r'* determinando el nivel de fortaleza del algoritmo AFGH, pudiendo ser en esta versión cualquiera de los valores 1, 2 o 3. Siendo de menor a mayor fortaleza.

Durante el proceso de ejecución es posible definir el comportamiento de dicho actor, tanto en el uso de una fortaleza concreta de AFGH como si se desea que dicho actor muestre mensajes de depuración y/o generar un fichero .csv con resultados concretos acerca del tiempo de generación de parámetros de curva elíptica y los generadores *'g'* y *'Z'*.

Al tratarse de rutas remotas de akka fijas, se le proporciona al generador las rutas de los otros dos actores fijos, los cuales son el Proxy y el Controlador.

Para la correcta ejecución del actor, se han desarrollado 4 clases java cuyas funciones son las siguientes:

- MainGenerador.java: En este fichero se incluye el método main; es la que inicia el proceso de ejecución del resto de clases y en la que el usuario define los parámetros de depuración y el nivel de fortaleza deseado; dichos parámetros se llevan al resto de clases java que intervienen en la ejecución.

- Generador.java: Tal y como fue mostrado en la Tabla 10, esta clase se encarga de invocar a la clase GeneradorActor con los parámetros de los números *'q'* y *'r'*, en función de la fortaleza elegida en el método main de la clase MainGenerador.

- GeneradorActor.java: En esta clase se encuentra el entorno de ejecución del actor principal que simula un generador de parámetros y pares de claves. Dispone de un constructor al que el usuario puede definirle si desea generar todos los parámetros desde cero o utilizar unos parámetros ya existentes; de esa forma podría crear los generadores *'g'* y *'Z'* de una forma más rápida ahorrándose la ejecución de crear unos parámetros nuevos desde cero si el usuario lo considera necesario.

Dicha clase, como todo actor, dispone de una función característica onReceive().

Este método se invoca cada vez que el actor generador reciba una petición de generación de claves por parte de otro actor. Observar como único parámetro de dicho método es de tipo Object, el cual debe ser categorizado con la palabra reservada 'instanceof' para determinar de qué tipo es y realizar la acción deseada. En este caso, debe ser un entero con valor 1, recibido desde un actor ajeno. Para contestar al actor, éste lo realiza con la función tell(...).

- GeneradorCSV.java: esta clase se encarga de recoger los datos de tiempo recibidos en un array en la clase GeneradorActor y procesarlos de tal forma que cree por salida un fichero .csv; dicho fichero puede importarse en Excel u otros programas de procesamiento de datos como Matlab o Wolfram Mathematica.

3.3.3.2 Bloque recifrador intermediario (o Proxy):

En este bloque se procede a mostrar el desarrollo del actor encargado de recibir claves de recifrado por parte de otros actores que se desean comunicar y de recibir mensajes cifrados para que éste aplique las operaciones de recifrado.

Se han definido para este caso las siguientes clases:

- MainProxy.java: esta clase, al igual que en el bloque generador, contiene el método principal de ejecución 'main', en este método se puede incluir la variable de depuración por si se desean obtener más mensajes acerca de los procesos que se realicen en todo momento, mostrándolos en consola.
- Proxy.java: esta clase tiene como única finalidad crear una instancia de la clase ProxyActor.

- ProxyActor.java: en esta clase se genera el actor que simula al bloque recifrador o Proxy. Este actor, como ya hemos comentado, tiene varias funciones: recibir claves de recifrado o recifrar mensajes, por ello, su método onReceive() debe realizar acciones concretas en función del tipo de objeto recibido.

Por ello, tenemos tres casos en dicho método: recibir claves del generador de claves, recibir una clave de recifrado o recibir un mensaje cifrado para aplicar la función para la que el proxy ha sido diseñado, re-cifrar dicho mensaje y enviarlo a su destinatario de la forma correcta para que éste sea capaz de descifrarlo.

3.3.3.3 Bloque Controlador de Objetos (o Destino):

En este bloque se procede a explicar el desarrollo del actor encargado de recibir los mensajes de todos los actores orígenes o sensores (véase como una estructura de red N a 1).

Este bloque contiene las siguientes clases:

- MainDestino.java: Esta clase, como el resto de bloques, contiene el método main encargado de crear instancias de la clase Destino.java.

- Destino.java: Esta clase se encarga de generar una instancia a la clase DestinoActor y definir las rutas akka (las cuales son fijas) del generador y del proxy.

- DestinoActor.java: Esta clase define el actor encargado de simular el controlador de objetos o, en el código denominado como "destino". En esta clase pueden darse cuatro casos: recibir par de claves del generador, recibir una petición de clave pública, recibir una clave pública de otro actor o recibir un mensaje cifrado.

El método onReceive() de este actor realiza una prueba de eco modificando valores del objeto SensorTemperatura y enviándolo de nuevo al actor remitente del mensaje. Con esto se consigue probar que el mensaje es capaz de transmitirse en ambos sentidos y, en caso de tener habilitada la variable de depuración correspondiente, generar tráfico de forma constante entre origen y destino con la finalidad de obtener tiempos de generación y de operaciones criptográficas.

3.3.3.4 Bloque de Objetos Inteligentes (u orígenes):

En este bloque se procede a mostrar el desarrollo de los actores encargados de simular objetos inteligentes, entre ellos, un sensor de temperatura.

Se pueden generar tantos actores de objetos inteligentes como queramos, pues akka asignará un número de puerto de carácter aleatorio para cada uno de ellos (ver configuración del fichero application.conf más adelante).

Para poder generar varios actores del mismo tipo es necesario invocar a la clase SensorActor tantas veces como se consideren necesarias.

Para este bloque, se han desarrollado tres ficheros de código fuente:

- MainSensor.java: esta clase contiene el método main necesario para ejecutar la aplicación que se encarga del bloque. En él se puede definir el número de actores que se necesitan nada más ejecutar dicho método; dicho valor será enviado como parámetro al generar la instancia de la clase Sensor.
- Sensor.java: en este actor se generan las instancias de la clase SensorActor.

- SensorActor.java: esta clase es la ejecuta el actor que simula el comportamiento de un objeto inteligente, entre ellos, un sensor de temperatura.

En este tipo de actores, se pueden tener cuatro tipo de situaciones: recepción de par de claves actualizado, petición de clave pública, solicitud de clave pública por parte de otro actor y recepción de un mensaje cifrado.

3.3.3.5 Otras clases imprescindibles utilizadas en los bloques:

Se puede observar el uso de un objeto de tipo CriptogramaAFGH en todos los actores excepto en el Generador, dicha clase es la que contiene un mensaje en sí pero ya cifrado, esté en la fase inicial (del origen al proxy) o en su fase final (del proxy al destino). Contiene todas las funciones necesarias para su correcto procesamiento en cualquiera de los bloques, se muestran algunas de ellas a continuación.

Los atributos de dicha clase son los siguientes:

- **ElementosTuple** criptograma: contiene el mensaje cifrado.

- **String** origen: contiene la dirección akka del actor origen.

- **String** destino: contiene la dirección akka del destino.

Otros objetos utilizados son los siguientes:

- ClavePublicaAjena.
- Claves.
- ElementosTuple.
- Peticion.

Todos ellos simplemente contienen objetos de tipo Element, en el caso de las claves y los elementos del Tuple excepto la clase Peticion que contiene además un entero indicando el tipo de petición que se desea realizar en el actor de destino.

Además, todas incluyen referencia a los actores origen y destino los cuales son comunicados.

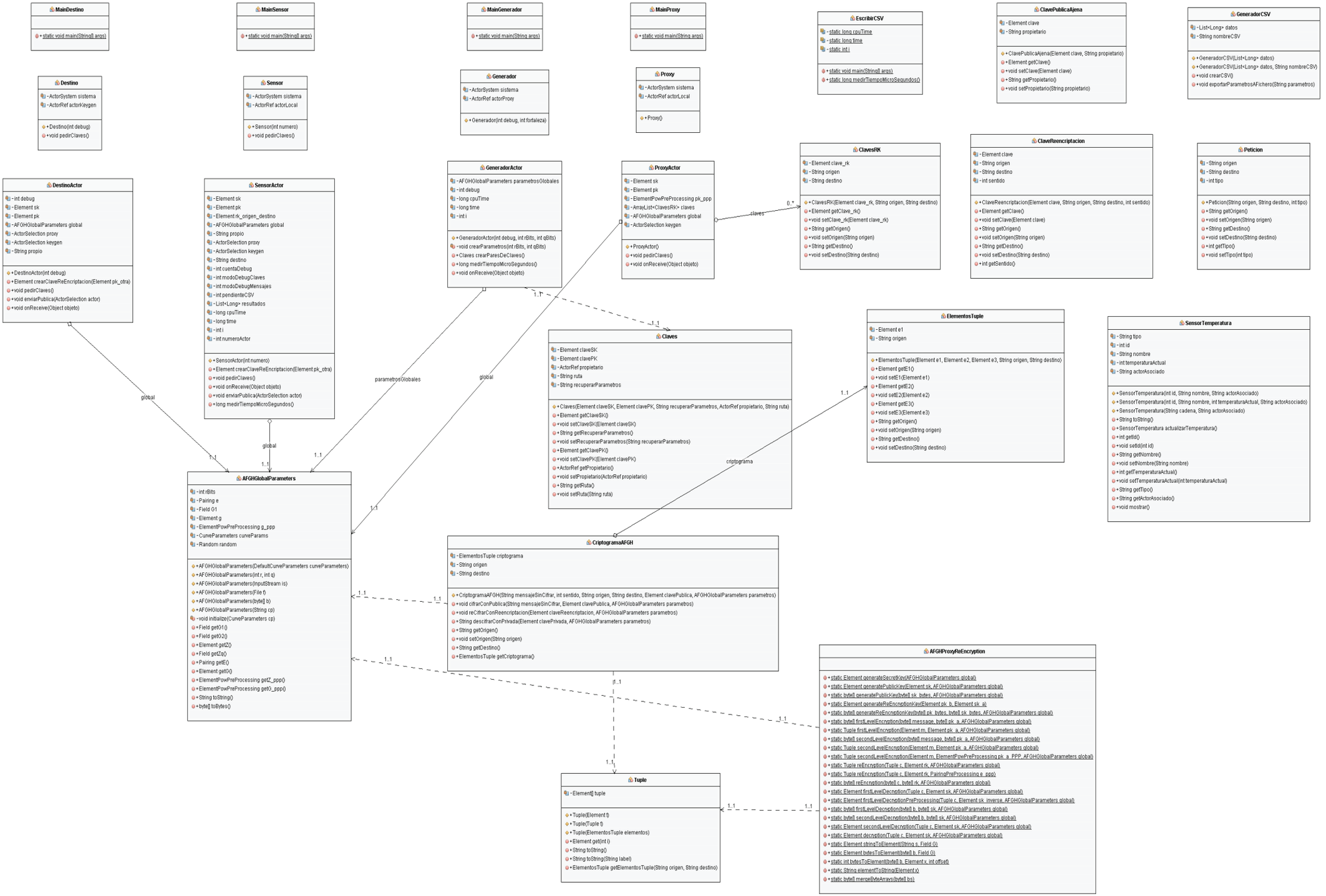


Figura 23. Diagrama de clases desarrolladas en este proyecto (en A3).

3.3.4 FICHERO DE CONFIGURACIÓN APPLICATION.CONF DE AKKA:

Este fichero es de esencial importancia para el correcto funcionamiento de todos y cada uno de los actores, además de otras librerías o funcionabilidades también utilizadas con Akka, como pueden ser los Serializadores.

Gracias a dicho fichero, Akka puede aplicar, entre otras, las siguientes configuraciones:

- Asignación de un serializador a una clase que en algún momento es necesaria la serialización; bien con el serializador predeterminado de Java o con el serializador de la librería Kryo for Akka.

- Asignación de un valor constante que ayuda al serializador Kryo a conocer qué clase es la que se ha serializado en todo momento, tanto en los actores origen y destino de dicha comunicación.

- Asignación de valores concretos de los puntos anteriores a cada actor por separado.

CAPÍTULO 4: PRUEBAS Y VALIDACIÓN

4.1 INTRODUCCIÓN

Una vez desarrollada la solución planteada sobre la capa extra de seguridad en entornos de red no confiables, utilizando como soporte el algoritmo AFGH por la librería Nics, se realizan las correspondientes pruebas para verificar el correcto funcionamiento tanto en la comunicación entre actores origen y destinos, tanto en el proceso de intercambio de claves públicas como en el proceso de cifrado, recifrado y descifrado de todos y cada uno de los mensajes.

Para ello se han utilizado variables de depuración que pueden ser habilitadas en cualquier momento en el propio código, el cual es posible importar en cualquier entorno de desarrollo (Netbeans, Eclipse...) de modo que se pueden realizar las verificaciones y modificaciones necesarias en el mismo para continuar avanzando en el futuro este tema del cual hablamos en este Trabajo Fin de Grado.

4.2 FUNCIONAMIENTO DEL SISTEMA COMPLETO

En este apartado se muestra la funcionalidad completa de los distintos bloques, los cuales están determinados por las siguientes direcciones remotas de Akka en nuestro entorno simulado:

- Actor generador de parámetros y pares de claves:
  “akka.tcp://SistemaKeygen@10.0.1.199:2202/user/ActorKeygen”

- Actor recifrador de mensajes o Proxy:
  “akka.tcp://SistemaProxy@10.0.1.201:2202/user/ActorProxy”

- Actor controlador de objetos o bloque de destino:
  “akka.tcp://SistemaDestino@10.0.1.202:2202/user/ActorDestino”

- Actores que simulan sensores (pueden ser varios con distintos puertos):
  “akka.tcp://SistemaSensor@10.0.1.200:PUERTO/user/ActorSensorX”

Nota: obsérvese como el puerto de los actores que simulan sensores de temperatura son valores variables, además del nombre del actor, que también es variable.

El primer bloque que se debe inicializar es el Keygen, para ello, tras ejecutar el fichero que contiene el método main(), el actor generador muestra la siguiente salida en consola, quedándose en un estado de reposo a la espera de peticiones entrantes de claves (nota, se ha realizado esta simulación con fortaleza 1):

SALIDA DE CONSOLA DEL ACTOR KEYGEN AL ARRANCAR POR PRIMERA VEZ
[INFO] [09/16/2015 15:40:02.331] [main] [Remoting] Starting remoting [INFO] [09/16/2015 15:40:02.659] [main] [Remoting] Remoting started; listening on addresses :[akka.tcp://SistemaKeygen@10.0.1.199:2202] [INFO] [09/16/2015 15:40:02.659] [main] [Remoting] Remoting now listens on addresses: [akka.tcp://SistemaKeygen@10.0.1.199:2202] [KEYGEN] Iniciado el keygen... esperando peticiones... [KEYGEN] Creando parámetros globales de curva elíptica... [KEYGEN] Parámetros de curva elíptica generados correctamente. [KEYGEN] A la espera de solicitudes de pares de claves por otros actores...

Tabla 10. Salida de consola del actor Keygen al arrancar por primera vez.

Tras ejecutar el generador, ya pueden ser iniciados el resto de bloques, se sugiere continuar por el Proxy, el cual, nada más iniciarse, intenta conectarse con el generador que ya ha sido arrancado; es en ese momento cuando la conexión resulta satisfactoria y el generador le envía un par de claves correspondiente:

SALIDA DE CONSOLA DEL ACTOR PROXY AL ARRANCAR POR PRIMERA VEZ
[INFO] [09/16/2015 15:43:52.450] [main] [Remoting] Starting remoting [INFO] [09/16/2015 15:43:52.731] [main] [Remoting] Remoting started; listening on addresses :[akka.tcp://SistemaProxy@10.0.1.201:2202] [INFO] [09/16/2015 15:43:52.731] [main] [Remoting] Remoting now listens on addresses: [akka.tcp://SistemaProxy@10.0.1.201:2202] [DEBUG-PROXY] Iniciado el proxy... a la espera de ordenes. CLAVES EN PROXY RECIBIDAS: Privada: 6188795191457128840278259434129435911760457433 78 Publica: 1278115539093775615704466037936588221905009237927808593985769151138216104288546567234776527205468188042616201411357011648599634398640517438901216097402347 0,1218459674655133645246812359412888778922692985865583430033141079647161816172169509730654568225581691024860920805057536798694296502677432397011183601375 5680,0 [PROXY] A la espera de recibir claves de reencriptación o mensajes...

Tabla 11. Salida de consola del actor Proxy al arrancar por primera vez.

El proxy, como ya se explicó anteriormente, puede recibir tanto peticiones de re-cifrado de mensajes entre extremos y, además, recibir las claves de re-encriptación necesarias para poder realizar dicha operación; recordemos que son necesarias dos claves de recifrado por extremos en comunicación, ya que el algoritmo AFGH es de carácter unidireccional y el Proxy no es capaz de calcular por sí mismo la clave de recifrado inversa hacia el sentido opuesto, puesto que en AFGH no se cumple dicha propiedad, al contrario que en BBS.

Una vez arrancados los actores Keygen y Proxy, procedemos a arrancar el actor que simula ser el controlador de objetos, éste nos muestra en consola la siguiente salida:

SALIDA DE CONSOLA DEL ACTOR CONTROLADOR AL ARRANCAR POR PRIMERA VEZ
[INFO] [09/16/2015 15:50:21.804] [main] [Remoting] Starting remoting [INFO] [09/16/2015 15:50:22.336] [main] [Remoting] Remoting started; listening on addresses :[akka.tcp://SistemaDestino@10.0.1.202:2202] [INFO] [09/16/2015 15:50:22.336] [main] [Remoting] Remoting now listens on addresses: [akka.tcp://SistemaDestino@10.0.1.202:2202] [DESTINO] Iniciado el destino... CLAVES RECIBIDAS EN DESTINO: Privada: 62989601983276170727048658067456647556267279989 9 Publica: 146996810929004033071894186782591316967603401542462687734358115445612912848632932385076759576623267462364447038566765450419337150469784631150964685981053357,11187392933886350189675813301859345155646529971069703788979698253491808625904993759780261978144143247337962238043966732433174206348581983640102936319807252,0 [DESTINO] A la espera de recibir mensajes para realizar acciones concretas...

Tabla 12. Salida de consola del actor Controlador al arrancar por primera vez.

Este actor puede recibir criptogramas ya recifrados, una clave pública de un actor remoto o una petición de su propia clave pública para ser enviada al actor remoto que la solicita.

Por último, podríamos arrancar uno o varios sensores de temperatura. Por simplicidad, iniciamos por ahora únicamente un sensor, mostrando en él la siguiente salida de consola:

SALIDA DE CONSOLA UN ACTOR SENSOR AL ARRANCAR POR PRIMERA VEZ
[INFO] [09/16/2015 15:53:02.680] [main] [Remoting] Starting remoting [SENSOR-TEMPERATURA] Iniciando un actor para un sensorTemperatura..... [INFO] [09/16/2015 15:53:03.039] [main] [Remoting] Remoting started; listening on addresses :[akka.tcp://SistemaSensor@10.0.1.200:49642] [INFO] [09/16/2015 15:53:03.039] [main] [Remoting] Remoting now listens on addresses: [akka.tcp://SistemaSensor@10.0.1.200:49642] CLAVES RECIBIDAS EN ORIGEN: Privada: 6325159366299327988748202176711526596212774425 25 Publica: 110556331062608910302110237113676457306474101119852878194727765558997572649135257210411920093422987294072655411246640781761112712916654344609362738248726 3,8234684227914248823741207230425879490734916454262730264856226845606652914924530890559697983108970714657298621919301990067566626889740232825013744885036837,0

Tabla 13. Salida de consola de un actor SensorTemperatura al arrancar por primera vez.

Para este tipo de actor, nada más arrancar, además de solicitar su par de claves pública y privada al generador, éste solicita al controlador su pública, para poder más adelante realizar una comunicación con el mismo.

SALIDA DE CONSOLA DE UN ACTOR SENSOR AL SOLICITAR UNA CLAVE PÚBLICA REMOTA
Recibida clave publica ajena con valor: PK: 14699681092900403307189418678259131696760340154246268773435811544561291284863 29323850767595766232674623644470385667654504193371504697846311509646859810535 7,11187392933886350189675813301859345155646529971069703788979698253491808625 9 04993759780261978144143247337962238043966732433174206348581983640102936319807 252,0

Tabla 14. Salida de consola de un actor SensorTemperatura al solicitar una clave pública remota.

También, automáticamente solicita el controlador al sensor su clave pública, procediendo a enviársela también (observar como coinciden los valores transmitidos)

SALIDA DE CONSOLA DEL ACTOR CONTROLADOR AL SOLICITAR UNA CLAVE PÚBLICA REMOTA
Enviando clave pública a origen con valor: PK_enviada: 14699681092900403307189418678259131696760340154246268773435811544561291284863 29323850767595766232674623644470385667654504193371504697846311509646859810535 7,11187392933886350189675813301859345155646529971069703788979698253491808625 9 04993759780261978144143247337962238043966732433174206348581983640102936319807 252,0 Recibida clave publica ajena con valor: PK_recibida: 11055633106260891030211023711367645730647410111985287819472776555899757264913 52572104119200934229872940726554112466407817611127129166543446093627382487263 ,82346842279142488237412072304258794907349164542627302648562268456066529149 24 53089055969798310897071465729862191930199006756662688974023282501374488503683 7,0

Tabla 15. Salida de consola del actor Controlador al solicitar una clave pública remota.

Una vez recibidas en ambos actores las claves públicas ajenas, proceden a enviar al proxy la clave de recifrado correspondiente para cada sentido:

SALIDA DE CONSOLA DEL ACTOR CONTROLADOR AL CREAR CLAVE DE RECIFRADO
Calculada clave de reencriptacion con valor: RK: 49447981353129061786189432496338139139746216674817432803259597708959279941575 72125510739645945168496205189898920660254575063921551446364592925213681755853 ,65654765205766327269175694445245785557912209167362046046633136014670996915 43 98028841630061577340913168899763589830187043695772854457764774643188251975339 2,0

Tabla 16. Salida de consola del actor Controlador al crear clave de Recifrado.

SALIDA DE CONSOLA DEL ACTOR SENSOR AL CREAR CLAVE DE RECIFRADO
Calculada clave de reencriptacion con valor: RK: 20314235191729073789388463500119859197963734380505704400766700162639048579232 67489048421534349642346394060593220737351631782221895942505176060443086314282 4,18164824454128785351530477920259941782157365761444696880170670206596321218 9 82148712526864625705147773562064620814481111938071181913531030856996032853991 850,0

Tabla 17. Salida de consola del actor SensorTemperatura al crear clave de Recifrado.

SALIDA DE CONSOLA DEL ACTOR PROXY AL RECIBIR AMBAS CLAVES DE RECIFRADO
Clave de reencriptacion recibida con valor RK: 20314235191729073789388463500119859197963734380505704400766700162639048579232 67489048421534349642346394060593220737351631782221895942505176060443086314282 4,18164824454128785351530477920259941782157365761444696880170670206596321218 9 82148712526864625705147773562064620814481111938071181913531030856996032853991 850,0 Clave de reencriptacion recibida con valor RK: 49447981353129061786189432496338139139746216674817432803259597708959279941575 72125510739645945168496205189898920660254575063921551446364592925213681755853 ,65654765205766327269175694445245785557912209167362046046633136014670996915 43 98028841630061577340913168899763589830187043695772854457764774643188251975339 2,0

Tabla 18. Salida de consola del actor Proxy al recibir ambas claves de Recifrado.

Que, como era previsto, las claves de re-cifrado son diferentes para cada sentido de la comunicación.

Supongamos que se desea enviar una cadena desde el sensor de temperatura hacia el controlador con esta estructura:

"SensorTemperatura|0|4|Sensor del salon|37"

Dicha cadena es el resultado de transformar de objeto a String la siguiente instancia:

Clase instanciada:	SensorTemperatura
Acción:	0 (nuevo valor)
Identificador:	4
Temperatura:	37 ºC

Para ello, el actor que simula ser el sensor cifra dicha cadena y la envía hacia el proxy, mostramos la salida de dicha operación:

SALIDA DE CONSOLA DEL ACTOR SENSOR AL CIFRAR UN MENSAJE
Se enviara informacion del sensor de temperatura: SensorTemperatura|0|4|Sensor del salon|37 [ORIGEN] CriptogramaAFGH Creado satisfactoriamente, enviando a PROXY... [ORIGEN] CriptogramaAFGH Enviado a PROXY satisfactoriamente.

Tabla 19. Salida de consola del actor SensorTemperatura al cifrar un mensaje.

Y, en el proxy, recibimos dicho criptograma C1 para transformarlo en el nuevo criptograma C2 que viajará hacia el controlador:

SALIDA DE CONSOLA DEL ACTOR PROXY AL RECIFRAR UN MENSAJE
[PROXY] CriptogramaAFGH Recibido en el proxy, recifrando... [PROXY] CriptogramaAFGH Recifrado y enviado a su destino.

Tabla 20. Salida de consola del actor Proxy al recifrar un mensaje.

En Destino, el controlador de objetos procede a descifrar con su clave pública el criptograma C2 recibido desde el Proxy:

SALIDA DE CONSOLA DEL ACTOR CONTROLADOR AL RECIBIR UN CRIPTOGRAMA RECIFRADO
Recibido un mensaje cifrado, descifrado con contenido: SensorTemperatura|0|4|Sensor del salon|37 Mostrando contenido del objeto reconstruido... DATOS DEL SENSOR DE TEMPERATURA #4 Nombre del sensor: Sensor del salon Temperatura actual: 37

Tabla 21. Salida de consola del actor Controlador al recibir un criptograma ya recifrado.

En el caso de que el proxy utilizase la clave de recifrado equivocada o fuese comprometido y, a consecuencia de ello enviase el criptograma a un destinatario que no es el deseado, el actor que recibe el criptograma C2 no es capaz de descifrar correctamente el mensaje, véase la siguiente imagen de ejemplo:

```
PK = 23755119552167169373272588506926464683581975976988150105380855294876454308365867999594164697636143
Recibida clave publica ajena con valor:
PK: 36131757326929760248575959182664264852182758649525425896894510384584105472471053992013951019640902
Calculada clave de reencriptacion con valor:
RK_a_b: 17756284182466029875567067276856919714335769622534056360575454522185162580529794341627133109110
Recibido un mensaje cifrado, descifrado con contenido:
SensorTemperatura|0|4|Sensor del salon|37
DATOS DEL SENSOR DE TEMPERATURA #4
Nombre del sensor:      Sensor del salon
Temperatura actual:     37
Recibido un mensaje cifrado, descifrado con contenido:
;□□□樓□@□□K~□□□□□□□□□υ拭□,□!f□"m¿□TOKQs□U#.lõ□□□^□□□W□□>□'□`□AlF□□f^u
□□M□#=□□        -□□□□□□□@□□□□£1□□□@□□*□□PV□D□□.u□.□□□z□□□
```

Tabla 22. Salida de consola del actor Controlador al recibir un mensaje ilegítimo o recifrado indebidamente.

Por lo que el algoritmo AFGH resulta funcional para el escenario planteado y funciona correctamente.

A continuación, en el siguiente apartado se muestran unas pruebas intensivas de rendimiento para cada sistema criptográfico, como por ejemplo el rendimiento a la hora de generar parámetros con diferentes tamaños de números primos, tamaños de clave, etc...

Esto es posible gracias a la importación de un fichero Java parte de la librería JavaCSV, concretamente descargada desde la web:
http://upcjorgerisco1.blogspot.com.es/

Esto nos permite introducir marcas de tiempo en la ejecución de nuestro código para que dichos resultados nos los imprima en un fichero .csv; dichos ficheros pueden ser abiertos porprogramas procesadores de cálculo numérico y de hojas de cálculo como pueden ser Matlab o Microsoft Excel. Para este proyecto utilizamos éste último como software de apoyo para la extracción de datos, muestra de gráficos con los resultados y agregando curvas de tendencia que nos proporcionará datos estadísticos, entre ellos el valor medio de tiempo dedicado para la ejecución de ciertas acciones.

4.3 PRUEBAS DE RENDIMIENTO

Como se mencionó en la Sección 3.2 del Capítulo 3, Planteamiento y Diseño, mencionamos tres tamaños diferentes de los números primos *'q'* y *'r'*, concretamente:

Número de bits de 'q'	Número de bits de 'r'
512 bits	**160 bits**
1536 bits	**256 bits**
7680 bits	**512 bits**

Tabla 23. Distintos tipos de fortaleza de cifrado para los números 'q' y 'r'.

Por lo que se procedió a definir en el actor generador de parámetros y claves, cada una de estas tres posibilidades, generando ambos números en su totalidad desde cero además de los parámetros *'g'* y *'Z'* correspondientes.

4.2.1 Generación de parámetros criptográficos

En este subapartado mediremos tiempos de ejecución de los tres tipos de soporte que nos proporciona el creador de la librería NiCS, apoyada en el modelo de actores de Akka, para comprobar el rendimiento en cada uno de los casos.

- **Generación de primos y parámetros realizadas con *q = 512 bits* y *r = 160 bits*.**

Se procede a calcular de forma iterativa durante 100 veces la generación de dichos números y a continuación los parámetros criptográficos, para así obtener unos valores más fiables teniendo en cuenta la carga del equipo donde se están realizando las pruebas, obteniendo los siguientes resultados:

Tiempos de generación (en ms), con q = 512 bits y r = 160 bits			
682,91	176,58	139,34	133,23
451,20	329,81	206,31	491,54
334,92	114,90	158,32	142,96
274,99	130,43	328,38	156,73
190,88	227,39	248,25	162,32
145,73	452,03	162,05	174,47
200,09	267,31	220,97	177,92
266,47	188,03	213,52	152,40
147,65	174,59	155,04	400,34
641,72	159,19	238,03	112,88

192,54	171,21	154,86	153,06
297,72	145,20	204,58	511,00
203,48	492,94	208,35	257,11
156,26	191,20	164,44	114,88
195,83	278,18	261,91	138,97
173,24	195,25	187,15	384,37
188,40	292,28	159,12	918,00
521,12	114,45	109,79	216,62
150,85	194,28	230,93	151,65
306,65	216,73	300,62	139,47
188,74	245,46	336,63	240,19
148,40	201,81	334,13	459,91
233,08	182,58	456,20	112,59
112,83	237,70	399,44	320,92
212,76	586,86	223,98	116,46

Tabla 24. Tiempos de generación de parámetros (en ms) en Fortaleza 1.

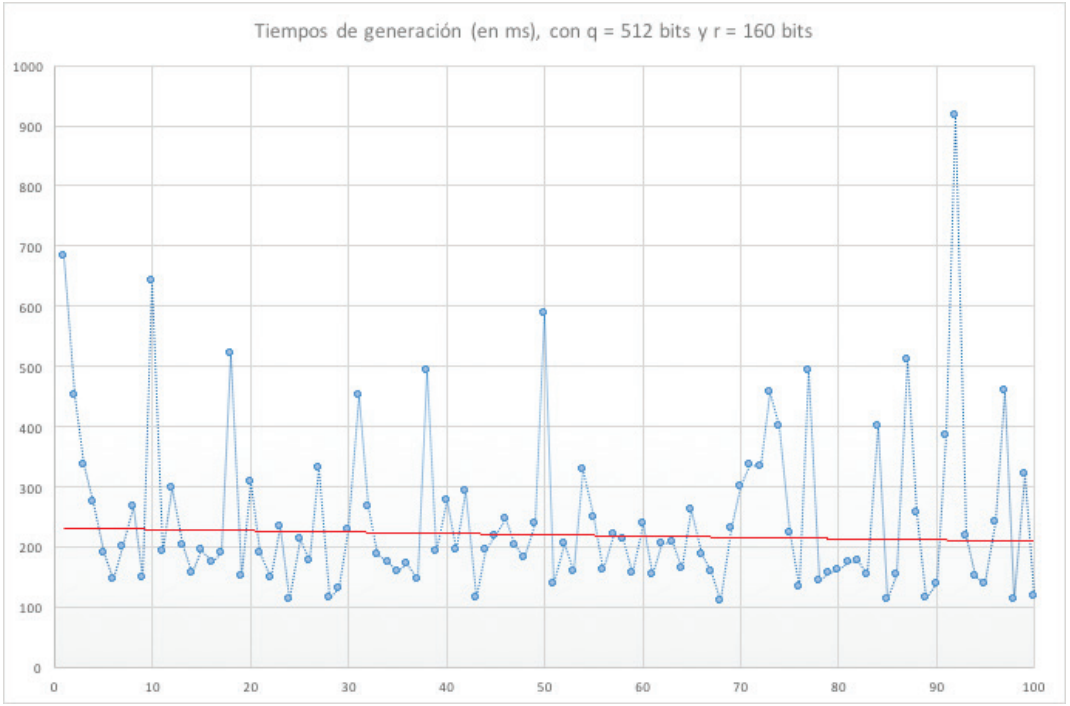


Figura 24. Tiempos de generación de parámetros (en ms) Fortaleza 1.

Observamos que, tras un análisis de dichos datos, se llega a la conclusión de que según la curva de tendencia, los tiempos tienden a estabilizarse en un valor concreto a la media, que, en dichos datos, está determinada en $t = 247{,}27\ ms$.

- **Generación de primos y parámetros realizadas con $q = 1536$ bits y $r = 256$ bits.**

Al igual que la prueba anterior, se vuelve a realizar una prueba iterativa de 100 generaciones de parámetros para comprobar el valor medio que necesita el sistema para generarlos:

Tiempos de generación (en segundos) con q=1536 bits y r=256 bits			
4,83	13,46	4,42	5,63
7,84	3,98	4,28	2,36
3,70	2,87	2,88	9,11
2,37	3,72	12,72	8,74
1,84	5,13	5,59	7,11
7,39	1,30	3,27	9,44
3,45	3,26	6,41	13,02
2,49	6,65	10,09	5,15
2,30	1,89	2,36	2,60
2,34	3,72	1,48	3,37
2,13	1,23	5,20	10,19
1,45	1,92	8,63	6,32
4,97	8,06	6,39	2,31
3,20	1,28	2,25	2,77
11,90	10,64	4,37	1,88
2,13	7,38	3,97	3,09
4,95	1,92	2,76	9,14
1,31	5,34	14,26	6,13
16,84	4,91	1,77	2,68
8,08	2,57	4,27	3,56
1,79	2,84	4,73	16,34
10,50	3,25	4,71	3,20
5,29	10,90	6,35	8,50
4,86	2,51	15,46	6,27
5,26	4,13	10,89	3,96

Tabla 25. Tiempos de generación de parámetros (en segundos) en Fortaleza 2.

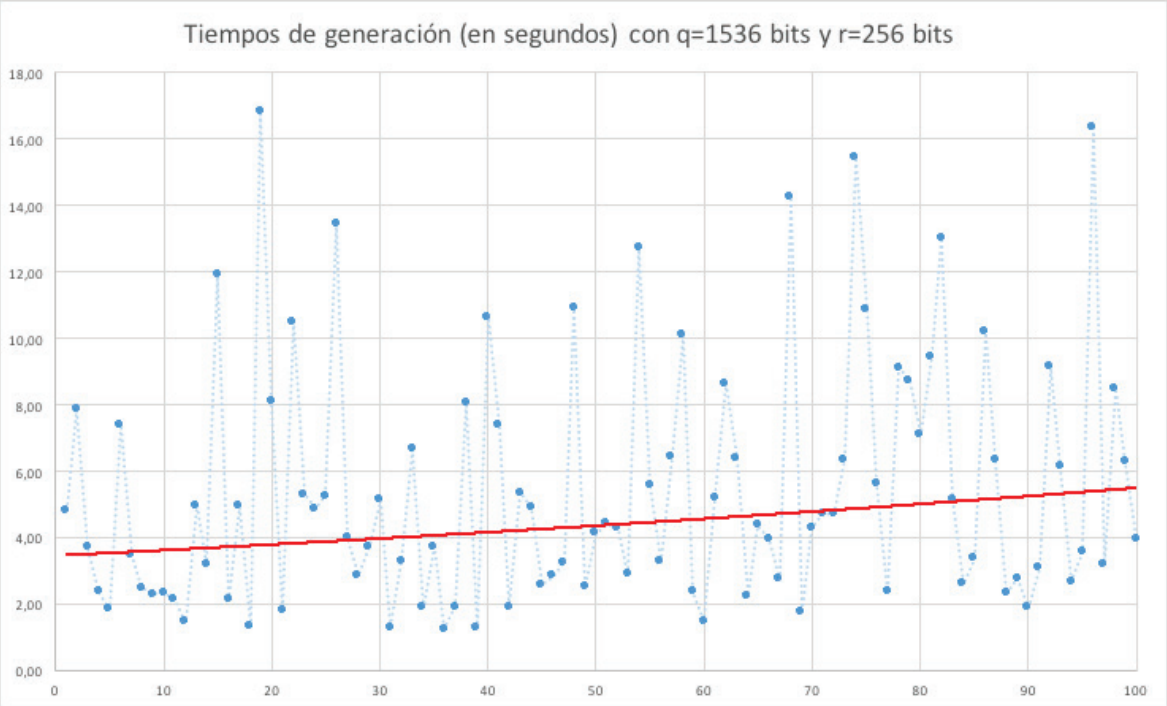


Figura 25. Tiempos de generación de parámetros (en segundos) Fortaleza 2.

En este caso, al analizar los datos, se puede determinar de que según la curva de tendencia (en color rojo), los tiempos tienden a incrementarse conforme avanza su ejecución. En esta configuración, el valor medio está determinado en $t = 5{,}40$ ***segundos***.

Dicho valor es un tiempo que se puede seguir considerando como aceptable, puesto que la generación de parámetros se realiza de una forma no tan habitual como la generación de claves.

- **Generación de primos y parámetros realizadas con $q = 7680$ *bits* y $r = 512$ *bits.***

Ahora se mostrarán los valores de los tiempos para el mayor tamaño de dichos números que nos sugiere el creador de la librería NiCS, se realizan también 100 iteraciones de generación de los mismos y de los parámetros:

Tiempos de generación (en minutos) con q=7680 bits y r=512 bits			
21,69	6,43	45,96	29,77
53,13	68,62	33,23	50,64
10,13	9,74	10,28	134,35
30,44	7,08	14,02	27,14
72,22	19,09	5,60	11,40

99,34	2,92	53,99	1,53
5,11	23,14	90,70	26,27
25,51	33,16	91,81	9,73
13,14	10,29	106,11	41,89
54,85	22,23	3,77	13,15
16,41	164,99	13,12	10,05
43,10	69,56	66,63	41,63
31,21	16,67	18,24	34,15
14,60	6,78	15,71	8,43
112,84	16,87	26,60	9,02
44,45	30,28	10,58	69,21
27,61	134,03	76,36	15,69
9,57	142,44	63,12	4,15
11,98	41,69	52,97	96,21
19,95	31,41	8,17	30,93
29,01	29,46	104,54	43,96
14,83	7,26	44,01	1,52
1,49	3,95	69,33	15,76
41,82	15,81	15,31	55,15
5,05	60,15	121,85	27,15

Tabla 26. Tiempos de generación de parámetros (en minutos) en Fortaleza 3.

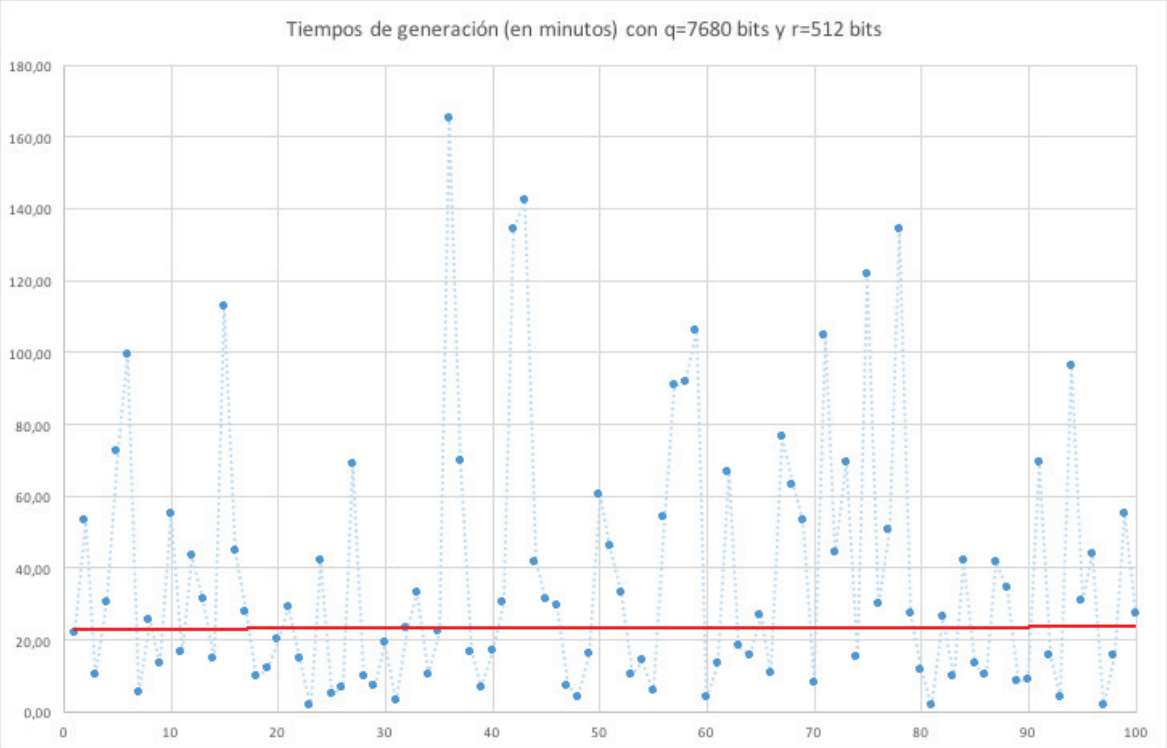


Figura 26. Tiempos de generación de parámetros (en minutos) Fortaleza 3.

En este caso, al analizar los datos, se puede determinar de que según la curva de tendencia (en color rojo), los tiempos tienden a incrementarse conforme avanza su ejecución pero de forma prácticamente estable. En esta configuración, el valor medio está determinado en $\mathbf{t = 37{,}54\ minutos}$.

Para este escenario, un tiempo medio de **37,54 minutos** hace que no sea práctico su uso, debido a que es un tiempo bastante alto de uso máximo de recursos de máquina e interrumpe durante todo ese tiempo la generación de claves, lo que se desaconseja su uso en un escenario de comunicación continua como el planteado.

No obstante, sí que sería posible en caso de que existiera una máquina adicional que exclusivamente se encargue de generar dichos parámetros para su posterior almacenamiento y uso por el actor Keygen, puesto que liberaría a la misma de este estado de bloqueo que los datos nos han proporcionado.

4.2.2 Generación de pares de claves

En este subapartado mediremos también tiempos de ejecución pero esta vez sobre el tiempo que dedica el generador para crear un nuevo par de claves a cada solicitante, obteniéndose los resultados siguientes para cada tamaño.

- **Generación de pares de claves *SK* y *PK* realizadas con $q = 512$ *bits* y $r = 160$ *bits*.**

En este caso, se han realizado 10.000 iteraciones puesto que se ha observado que el tiempo de cómputo no es demasiado alto, por ello, los resultados de este caso los mostramos únicamente mediante gráfica debido a la alta cantidad de muestras de tiempo (en milisegundos):

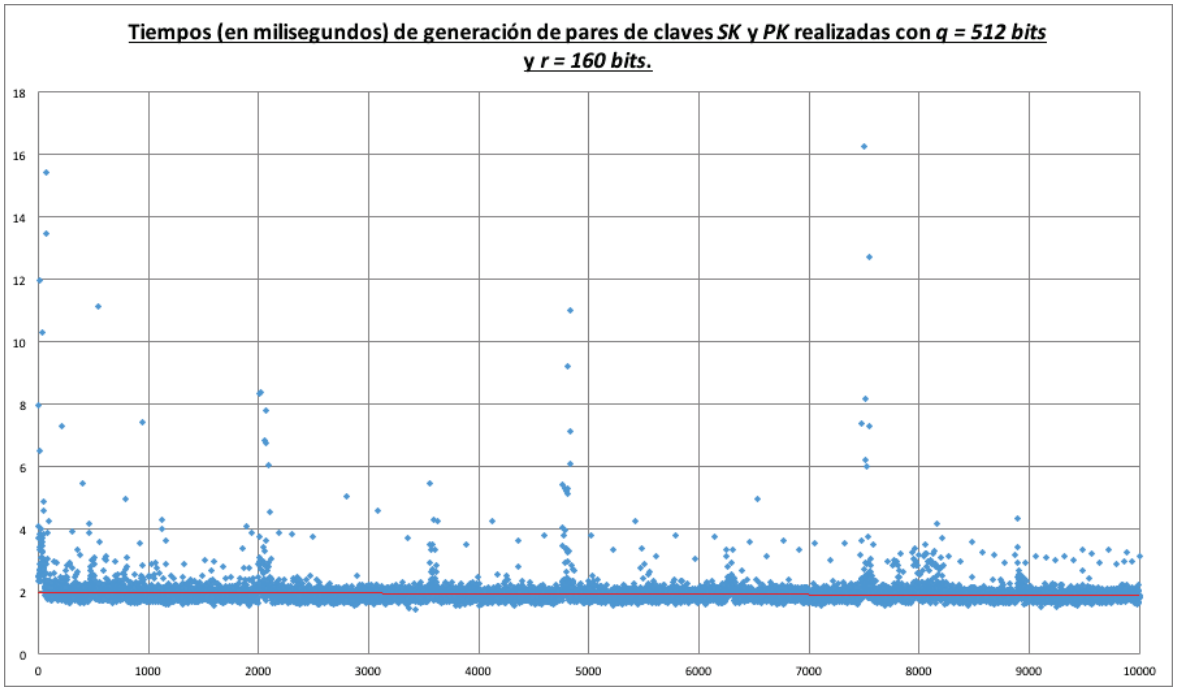


Figura 27. Tiempos de generación de pares de claves (en ms) Fortaleza 1.

Según los datos observados en la gráfica y en la curva de tendencia, que salvo cargas puntuales en el actor, los valores de generación tienden a estar concentrados respecto a la media, situada en $t = 1{,}94$ **milisegundos**.

- **Generación de pares de claves *SK* y *PK* realizadas con *q = 1536 bits* y *r = 256 bits.***

En este otro caso, se han realizado tan solo 100 iteraciones puesto que se ha observado que el tiempo de cómputo es mayor que en el caso anterior, por ello, los resultados de este caso los mostramos tanto en tabla como en un gráfico (en milisegundos):

Tiempos de generación (en ms), con q = 1536 bits y r = 256 bits			
20,70	16,43	17,70	16,74
23,01	16,29	17,07	16,13
17,11	16,44	17,76	21,31
16,80	16,00	17,72	16,59
17,65	16,83	17,07	16,25
18,02	16,75	18,23	17,98
17,61	16,73	18,00	16,04
17,15	17,74	16,82	16,65
17,18	17,23	17,69	17,42
15,77	16,41	16,81	15,90
16,59	16,27	15,62	17,34
28,49	16,35	17,76	16,71
15,61	16,22	16,46	16,00
17,12	17,25	17,62	16,85
16,97	16,49	16,56	16,90
16,63	17,89	17,29	16,61
16,80	18,18	16,81	15,00
17,27	17,09	18,82	17,73
16,12	16,49	17,44	16,15
16,56	17,01	32,24	16,06
17,15	22,71	17,12	17,52
17,17	18,40	18,09	15,70
16,28	17,29	16,49	17,64
17,02	16,61	17,69	15,99
15,76	17,78	17,34	16,63

Tabla 27. Tiempos de generación de pares de claves (en ms) en Fortaleza 2.

Según estos resultados, obtenemos que el valor de la media está situado en este caso en ***t = 17,39*** **milisegundos**, lo cual nos sigue dando un valor dentro de lo factible para este escenario.

Se muestra a continuación la gráfica resultante de los tiempos de generación de claves públicas y privadas a lo largo del tiempo durante 100 iteraciones, observamos que la tendencia en dichos tiempos, también salvo casos puntuales de sobrecarga en el sistema del actor, es en valores cercanos a la media:

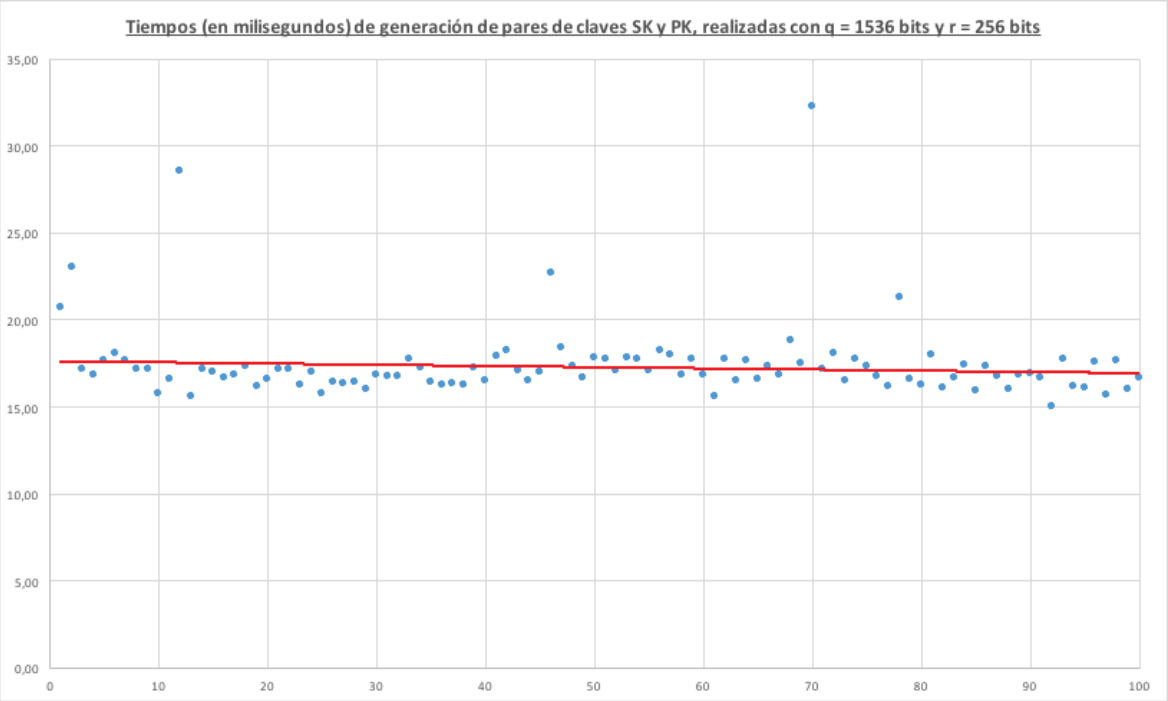


Figura 28. Tiempos de generación de pares de claves (en ms) Fortaleza 2.

Generación de pares de claves *SK* y *PK* realizadas con $q = 7680$ *bits* y $r = 512$ *bits.*

En el caso de mayor fortaleza en cuanto al tamaño de los parámetros generadores y del tamaño de las claves, se han realizado también las pruebas iterativas con 100 resultados, mostrando los tiempos de generación en la tabla:

Tiempos de generación (en ms), con q = 7680 bits y r = 512 bits			
546,35	548,65	568,88	531,42
554,63	542,65	630,33	548,55
597,97	551,80	620,85	647,24
557,86	555,01	579,39	546,72
552,50	574,44	569,83	536,74
549,46	572,68	542,63	580,89
525,84	639,58	543,98	552,64
555,27	584,25	555,76	562,04
544,81	564,96	647,37	552,88
543,34	547,34	554,69	550,76

570,23	563,50	593,80	546,77
579,20	552,48	587,95	647,79
554,86	545,53	540,23	561,90
548,41	580,83	568,84	551,79
545,91	597,13	555,44	569,37
551,76	672,52	539,97	549,04
619,69	575,02	524,27	559,41
532,27	558,23	593,23	549,80
547,67	568,56	539,46	552,20
520,57	580,46	529,37	547,51
562,28	568,71	568,62	624,59
553,88	566,84	535,32	564,51
592,48	561,50	535,16	583,47
568,42	588,08	564,60	595,21
534,81	657,02	573,88	544,91

Tabla 28. Tiempos de generación de pares de claves (en ms) en Fortaleza 3.

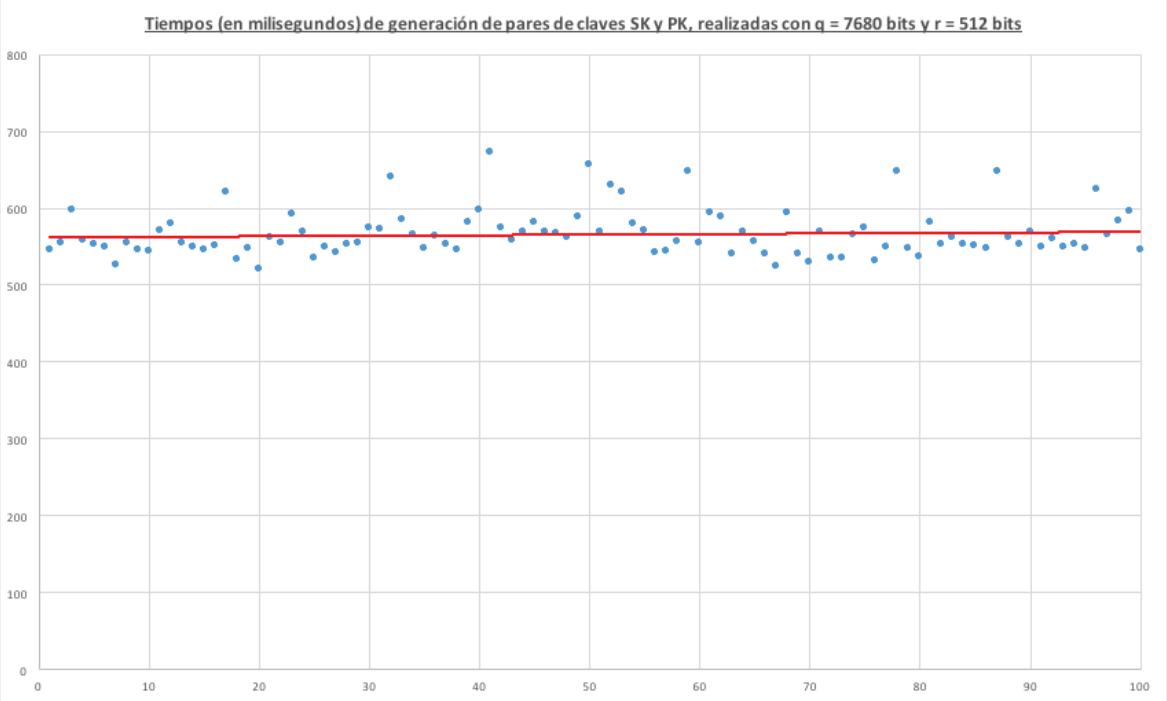


Figura 29. Tiempos de generación de pares de claves (en ms) Fortaleza 3.

Podemos observar que, respecto a los dos casos anteriores, a mayor tamaño de clave el tiempo de cómputo aumenta con un comportamiento exponencial, la media en esta prueba se ha situado en $t = 566{,}50$ **milisegundos**. Valor que no parece demasiado excesivo e incluso factible, pero habría que observar la capacidad de cálculo suponiendo un sistema con una gran cantidad de actores realizando peticiones simultáneamente y que recordamos, como ya hemos visto anteriormente,

todas las peticiones de claves quedan pendientes de atender según órden de llegada, lo cual es un sistema con espera.

- **Conclusión acerca de los tiempos obtenidos para cada tamaño de clave (mayor fortaleza cuanto mayor tamaño de clave)**

En los resultados anteriores hemos obtenido los siguientes valores medios acerca de los tiempos de generación de los pares de claves:

$t_1 = 1{,}94 \ ms$	*para la fortaleza 1:*	$q = 512$ *bits*	$r = 160$ *bits*
$t_2 = 17{,}39 \ ms$	*para la fortaleza 2:*	$q = 1536$ *bits*	$r = 256$ *bits*
$t_3 = 566{,}50 \ ms$	*para la fortaleza 3:*	$q = 7680$ *bits*	$r = 512$ *bits*

Lo que nos lleva lugar a deducir que en función de la seguridad que se desee en la comunicación sería posible elegir cualquiera de las tres, puesto que todos los valores están dentro de unos tiempos de espera coherentes, aunque habría que analizar el compromiso seguridad-velocidad, puesto que en ciertos entornos en los que se pueda cambiar de parámetros generadores regularmente sería suficiente utilizar el tipo de fortaleza 1, en cambio, para sistemas de seguridad media lo ideal sería la utilización del algoritmo AFGH utilizando como configuración la fortaleza de claves 2.

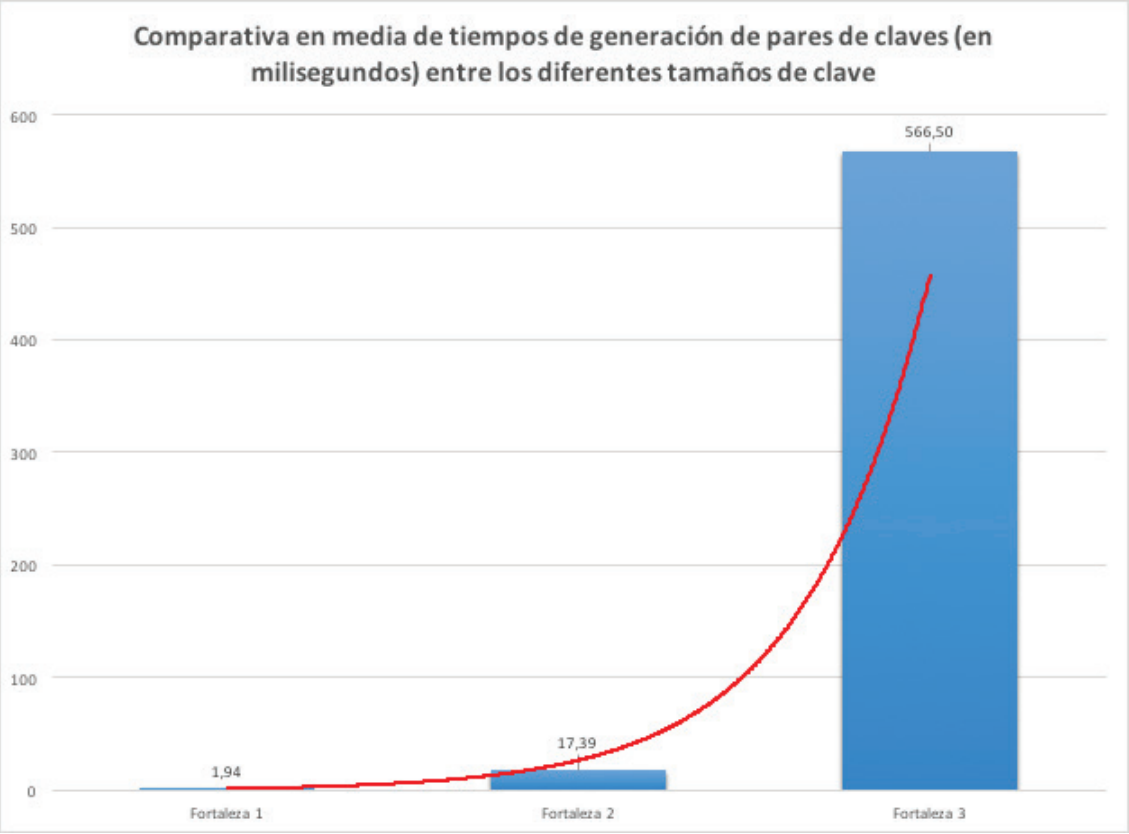


Figura 30. Comparativa de tiempos de generación de pares de claves (en ms).

Como se ha mencionado, se puede observar en el gráfico de la Figura 24 que, a mayor tamaño de parámetros y por lo tanto de claves, el tiempo de cómputo crece con carácter exponencial.

Quedaría demostrada dicha tendencia creciente respecto al tiempo de generación y cómputo de cualquier acción criptográfica; si se utilizan números más grandes (con mayor número de bits) requerirá de la aplicación un mayor tiempo de cómputo.

4.2.3 Envío y recepción de mensajes cifrados entre actores

En este subapartado mediremos los tiempos de ejecución que han necesitado los tres actores implicados en la generación, cifrado, recifrado, descifrado y regeneración de un mensaje conteniendo datos de un sensor de temperatura.

Los mensajes enviados / recibidos en estas pruebas son cadenas cifradas con esta construcción: "SensorTemperatura|0|ID|Nombre del sensor|TEMP EN ºC".

Por ello, vamos también a mostrar los resultados para los tres tipos de fortaleza que hemos visto hasta ahora.

- **Envío y recepción de mensajes cifrados con $q = 512$ bits y $r = 160$ bits.**

En este caso, se han realizado 1.000 iteraciones y debido a ello, solo mostramos los valores de dichos resultados en la siguiente gráfica, además de la línea de tendencia en color rojo:

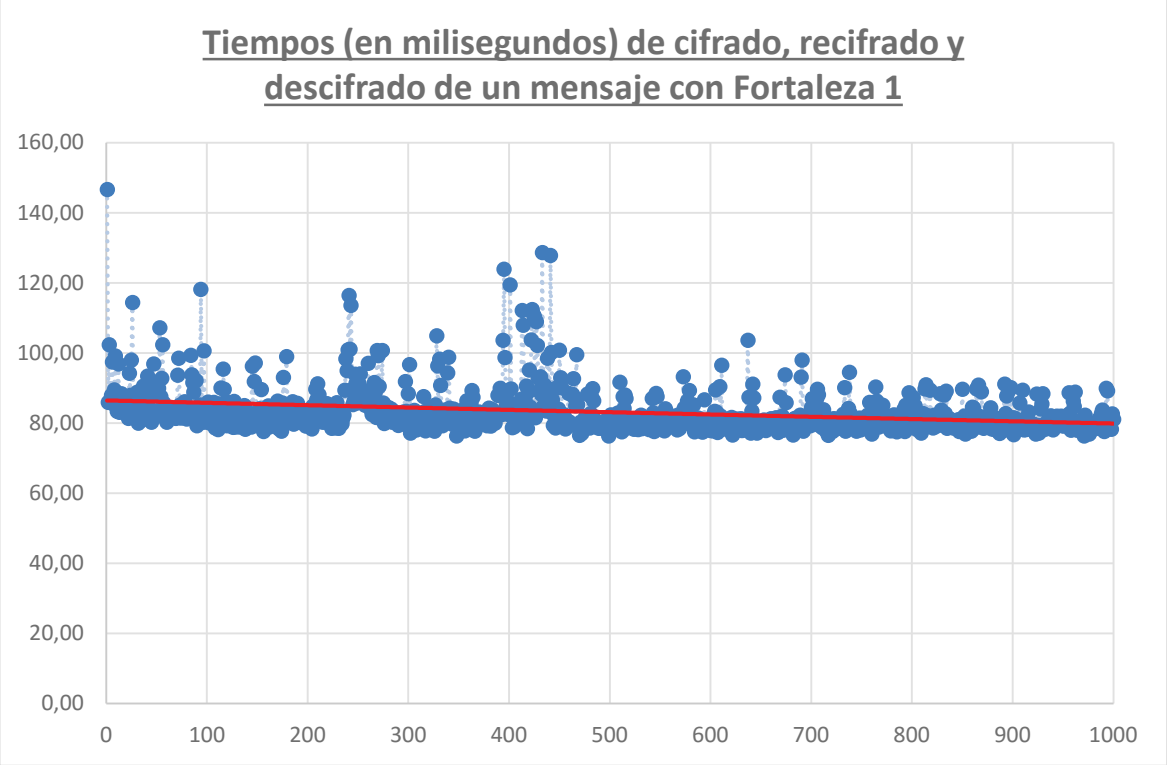


Figura 31. Tiempos de cifrado, recifrado y descifrado con Fortaleza 1 (en ms).

Como podemos observar, es cierto que al inicio los tiempos son un poco mayores, pero a medida que incrementan las iteraciones la tendencia es que vayan bajando hasta estabilizarse, salvo resultados de tiempo más altos entre las 200-300

iteraciones, ya que se ha producido un pico de carga en el sistema.

El valor medio de tiempo de procesamiento en este caso es de $t = 113{,}86\ ms$. Dicho valor se considera aceptable para la transmisión contínua de información.

- **Envío y recepción de mensajes cifrados con $q = 1536\ bits$ y $r = 256\ bits$.**

En este caso, también se han realizado 1.000 iteraciones y por consecuencia, como en el caso anterior, solo mostramos los valores de dichos resultados en la siguiente gráfica, además de la línea de tendencia en color rojo:

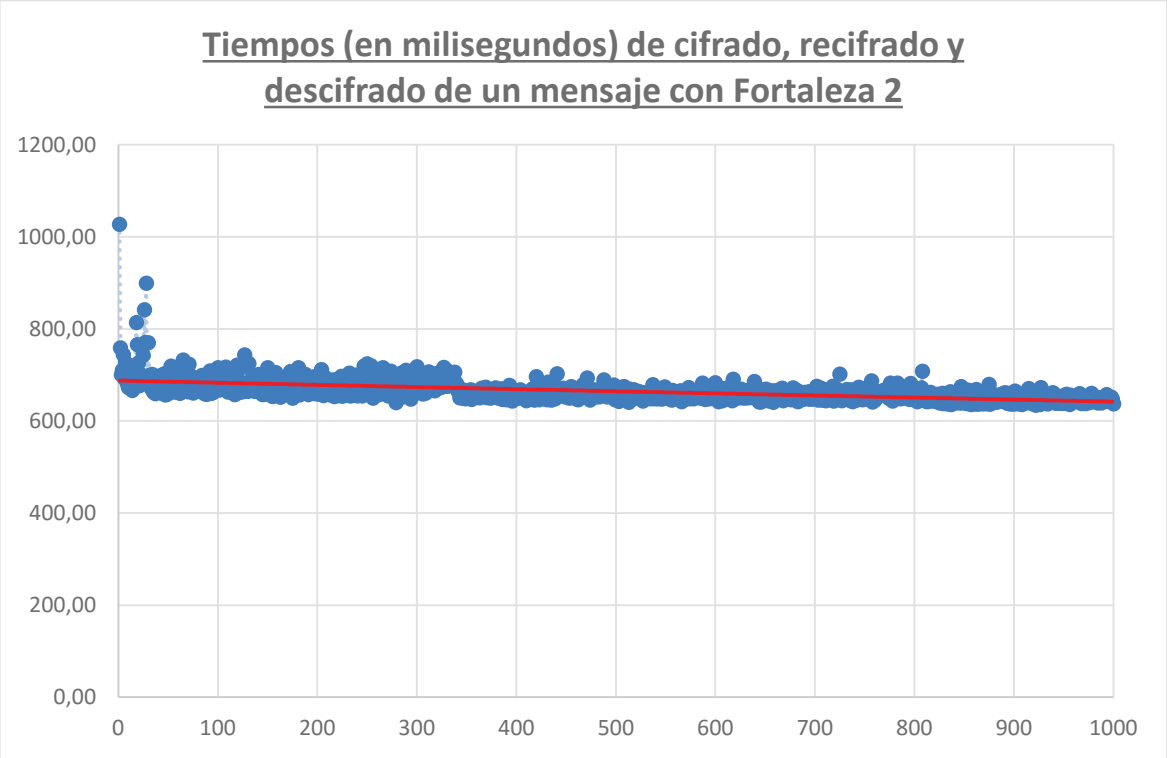


Figura 32. Tiempos de cifrado, recifrado y descifrado con Fortaleza 2 (en ms).

Es posible observar también un comportamiento decreciente o estabilizador en cuanto al tiempo necesario para realizar dichas tareas criptográficas y de envío.

El valor medio de tiempo de procesamiento incluyendo las tareas de cifrado en origen, recifrado en proxy y descifrado en el controlador, incluyendo transmisiones este caso es de $t = 665{,}05\ ms$.

Se puede decir que este resultado sería un tiempo computacional un poco alto pero podría ser útil en sistemas que no envíen información constantemente, porque de

utilizarlo en entornos de envío contínuos podría producir cuello de botella en el proxy.

- **Envío y recepción de mensajes cifrados con $q = 7680$ bits y $r = 512$ bits.**

Para esta última prueba con tiempos de procesamiento criptográfico y de envío sobre mensajes, solo se ha realizado el experimento sobre 100 mensajes debido al alto tiempo que conlleva realizarlos, por ello en este caso sí que mostramos los resultados en una tabla y, a continuación, mostramos los valores de dichos resultados en la posterior gráfica, además de la línea de tendencia en color rojo:

Tiempos de procesamiento (en ms), con q = 7680 bits y r = 512 bits			
26,24	23,07	24,64	25,46
22,67	22,71	23,48	24,90
22,26	23,28	23,21	26,77
22,47	22,81	23,06	23,19
22,26	22,58	24,01	23,86
22,02	22,52	22,92	23,90
22,33	26,63	26,69	23,31
21,64	22,94	24,91	22,91
22,13	23,93	24,68	25,32
22,12	23,40	23,37	27,66
21,95	22,92	22,37	26,62
23,44	23,88	22,37	27,17
23,08	23,50	22,39	26,57
24,80	22,85	23,34	27,16
22,88	22,87	22,53	24,68
25,08	23,50	22,71	24,13
24,48	23,53	22,55	22,97
21,45	24,10	23,71	22,90
22,49	23,94	23,94	24,17
22,52	23,63	23,75	23,35
23,94	23,14	22,59	24,69
23,54	23,49	24,69	22,57
23,34	22,89	26,61	22,54
23,02	24,68	25,86	22,61
22,89	25,67	24,11	22,51

Tabla 29. Tiempos de operación de cifrado/recifrado/descifrado de mensajes (en ms) en Fortaleza 3.

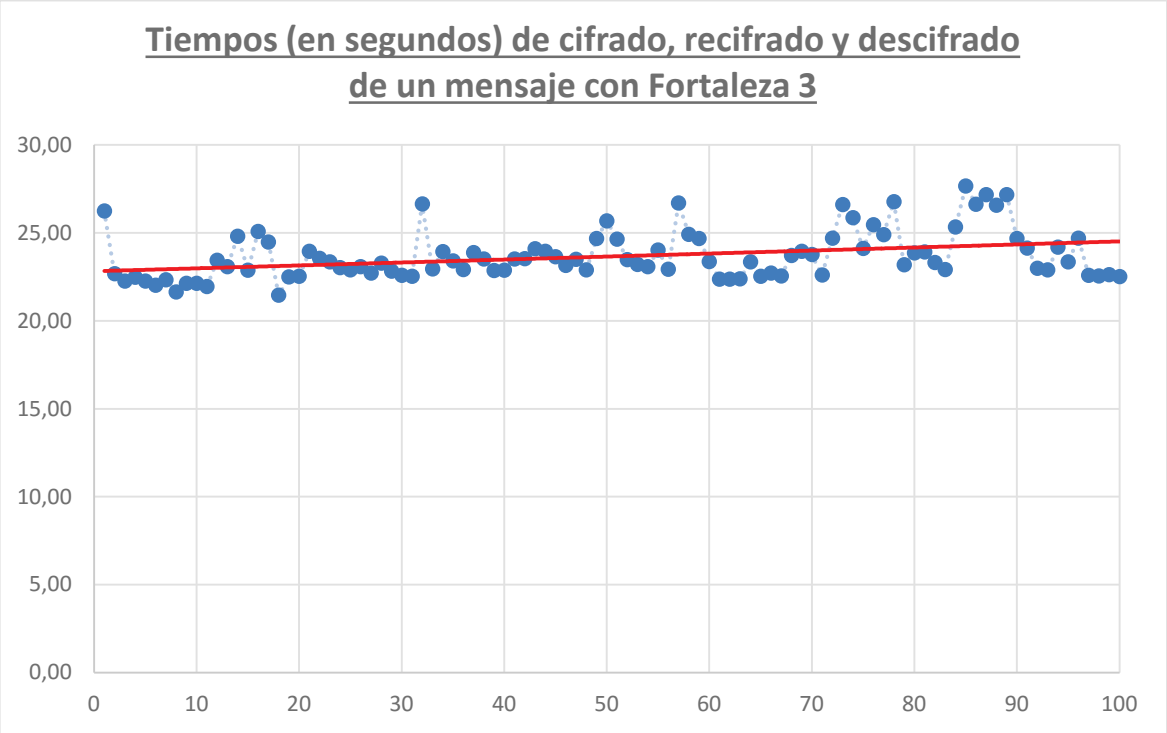


Figura 33. Tiempos de cifrado, recifrado y descifrado con Fortaleza 3 (en segundos).

En este último caso observamos que la línea de tendencia en este caso es con pendiente positiva, lo que indica que el sistema tiene cierta inestabilidad cuanto mayor sea su actividad en procesamiento de mensajes, luego indica un punto negativo respecto al uso de esta fortaleza del algoritmo.

El valor medio de tiempo de procesamiento incluyendo las tareas de cifrado en origen, recifrado en proxy y descifrado en el controlador, incluyendo transmisiones este caso es de $\boldsymbol{t = 23{,}70\ segundos}$.

Este resultado nos proporciona un tiempo de procesamiento demasiado alto para una transmisión constante de mensajes, por lo que se desaconseja su uso en entornos en los que la comunicación deba ser de carácter continuo.

Además, no es el único inconveniente de esta fortaleza, sino que también ya obtuvimos en las pruebas anteriores los altos tiempos de generación de parámetros criptográficos, que en el momento de la ejecución de esta prueba tardó en realizarse 45 minutos además de los tiempos de generación de claves y del procesamiento de los mensajes mostrados en la tabla anterior.

4.4 CONCLUSIÓN

4.3.1 Conclusión

Finalizado el trabajo y tras la ejecución de las pruebas de rendimiento y validación se comprueba que el sistema funciona acorde a las especificaciones planteadas en el diseño y nos proporciona los resultados esperados; se han analizando cada una de las posibles soluciones criptográficas propuestas en el diseño como los tiempos de procesamiento de cada una de ellas.

Tras realizar pruebas de envío y recepción de mensajes con diferente contenido cifrado, todos los bloques cumplen con el propósito planteado, el cual trataba de mantener segura una comunicación mediante cifrado punto a punto de manera asíncrona gracias a la utilización de una entidad intermediaria (proxy) independientemente de las redes (confiables o no) que atraviesen los datos.

En cuanto a los resultados de las pruebas de rendimiento y validación anteriores, disponemos de los siguientes valores finales en unidades de tiempo:

PRUEBAS	Con fortaleza 1	Con fortaleza 2	Con fortaleza 3
Generación de parámetros	247,27 ms	5,40 seg	37,54 min
Generación de pares de claves	1,94 ms	17,39 ms	566,50 ms
Procesamiento, envío y recepción de mensajes	113,86 ms	665,05 ms	23,70 seg

Tabla 30. Compilación de tiempos obtenidos en todas las pruebas de validación.

Por un lado, tras los análisis de los datos proporcionados por los actores en formato .csv podemos llegar a la conclusión de que la fortaleza del algoritmo AFGH recomendada para este escenario es la denotada como 'Fortaleza 2', con un tamaño de *'q' = 1536 bits* y *'r' = 512 bits* debido a que la que mejor relación entre seguridad y tiempo de procesamiento nos proporciona.

Aunque para situaciones en las que se genera una gran cantidad de tráfico de forma simultánea y constante, si se renuevan periodicamente las claves no existiría ningún tipo de inconveniente en utilizar el nivel de seguridad que proporciona la configuración de 'Fortaleza 1'. Salvo casos muy puntuales en los que se envíe información con poca frecuencia pero sea de especial sensibilidad, se podría utilizar el tercer grado de fortaleza del algoritmo.

En cuanto a las dificultades más significativas que se han presentado en el desarrollo de la solución propuesta en este Trabajo de Fin de Grado han sido relacionadas con la unión de las diferentes tecnologías utilizadas. Todas ellas proporcionadas gracias a las librerías a las que se hizo referencia en el apartado 3.3.

Inicialmente la generación de los parámetros se realizaba exclusivamente en el actor encargado de la generación de parámetros (Keygen), pero se daba el inconveniente de que los procesos de cifrado y descifrado debían de realizarse también en dicho actor; debido a ello se procedió a realizar un cambio de diseño en el proceso de petición/generación de claves. En el diseño final del Keygen se decidió que en el momento en el que se envíe el par de claves al actor que lo solicite, también se incluya una cadena con valores resumidos que determinan a los parámetros generados; de esta forma se pueden "regenerar" en el resto de actores sin necesidad de calcular de nuevo los números *'q'* y *'r'*. Este proceso es necesario porque son necesarios los generadores *'g'* y *'Z'* en el proceso de cifrar, recifrar y descifrar.

Más adelante se presentó otro problema, y es que inicialmente se utilizaba el serializador por defecto de Java; éste no es capaz de serializar clases ya precompiladas, procediendo a la utilización del serializador Kryo.

Por último, y es por ello la consecuencia de enviar como cadena los datos acerca de los parámetros junto al par de claves a los actores, el serializador Kryo entraba en bucle al intentar serializar una instancia del tipo AFGHGlobalParameters, además de las relacionadas con el objeto Tuple. Por ello para el primer caso se realizó la decisión de reconstruir en destino los parámetros y, en el segundo caso, se procedió a la separación de los elementos que componen el Tuple (el cual venía en formato de Array) en varios elementos independientes, ya que para este trabajo siempre se componen de tres elementos, por ello se crea este nuevo objeto que sí se puede serializar en Kryo, el cual se denomina con el nombre de clase 'ElementosTuple'.

Este proyecto me ha motivado a conocer con más detalle los procesos por los que pasan los mensajes mientras atraviesan varios procesos de cifrado, además del trabajo con un entorno de desarrollo que anteriormente no había utilizado (Netbeans) y la unión de varias funcionalidades diferentes en un mismo proyecto trabajando con librerías de Java de terceros importadas, especialmente la facilidad con la que se pueden realizar sistemas complejos, asíncronos y con espera gracias al modelado de actores que implementa la librería Akka.

4.3.2 Líneas Futuras de Trabajo

En cuanto a las líneas futuras sobre este Proyecto cabe mencionar que dispone de un margen de ampliación muy amplio. En este apartado se proponen varias de las posibles opciones que podrían ser mejora del escenario planteado en el mismo.

Se podría diseñar un sistema completo más complejo, incluyendo además de sensores de temperatura, elementos dependiendes de otros sistemas más grandes, o incluso simulación de dispositivos móviles enviando y recibiendo información de una Crowd Sensing real, conectando el actor "Controlador de Objetos" a un nuevo actor encargado de gestionar una base de datos MySQL de forma que dichos dispositivos móviles puedan enviar peticiones de contribución de datos o de recepción de los que ya existen en la BD.

Otra mejora a agregar podría ser la implementación de otros algoritmos criptográficos con la finalidad de realizar pruebas en los mismos, para comprobar su eficiencia en un modelo de actores que simulan elementos de la Internet of Things; un ejemplo podría ser la implementación del algoritmo BBS [25], el cual ya fue mencionado en el apartado 2.5 Proxy Re-Encryption, u otros más complejos basados en la misma filosofía. De esa forma podríamos obtener un mayor número de resultados que comparar con la finalidad de comprobar cuál es más útil en este entorno.

En cuanto a posibles modificaciones en las tecnologías utilizadas una vez finalizado el proyecto, se podría haber omitido el uso del Serializador de Java por defecto desde el principio, de forma que se utilice únicamente el importado por la librería Kryo para Akka, ya que la escalabilidad de este último es incomparable con la que nos puede proporcionar el serializador por defecto de Java, especialmente la posibilidad de serializar prácticamente cualquier objeto.

APÉNDICE: ORGANIZACIÓN

AP.1: MARCO REGULADOR

En cuanto al marco regulatorio, tanto los sistemas Crowd Sensing actuales que están en producción como cualquier otro sistema de la Internet of Things futuro tienen como finalidad enviar datos a través de una o varias redes. Dichas redes no necesariamente tienen por qué ser totalmente confiables, ya sea una red de conexión permanente debido a que se trate de un elemento fijo y conectado mediante Ethernet; o el uso de varias redes ya que estos dispositivos pueden ser móviles y por ello se conectarán a redes de forma inalámbrica.

La información transmitida desde dispositivos recolectores de datos para un uso bien local o colectivo de los mismos, hace que dichos datos sean de carácter sensible y personales, luego es necesario que los paquetes de datos que contengan información personal viajen con una capa de seguridad de fortaleza considerable para prevenir la usurpación por terceros de dicha información o poner en riesgo la privacidad de los individuos propietarios de los dispositivos que se encuentran interconectados.

De no disponer de medidas de seguridad con dicha fortaleza, podría ir en contra de una o varias leyes locales de algunos países, como en España la Ley Orgánica de Protección de Datos (LOPD) o ir en contra de normativas de protección de datos de la Unión Europea por una violación de la intimidad en caso de no realizar dichas comunicaciones correctamente cifradas.

Sobre la aplicación de regulaciones sobre Crowd Sensing, en la actualidad no existe ninguna regulación sobre el número de dispositivos máximos que pueden actuar de semillas para dicho sistema ni existe ninguna normativa acerca de un uso normalizado para esta tecnología, aunque puede que en un futuro sí que existan y acabe regulándose el uso de estos elementos.

AP.2: PLANIFICACIÓN DEL PROYECTO

Para la planificación, se ha construido el siguiente diagrama de gantt realizado con la utilidad Gantt Project y el software Microsoft Project 2013:

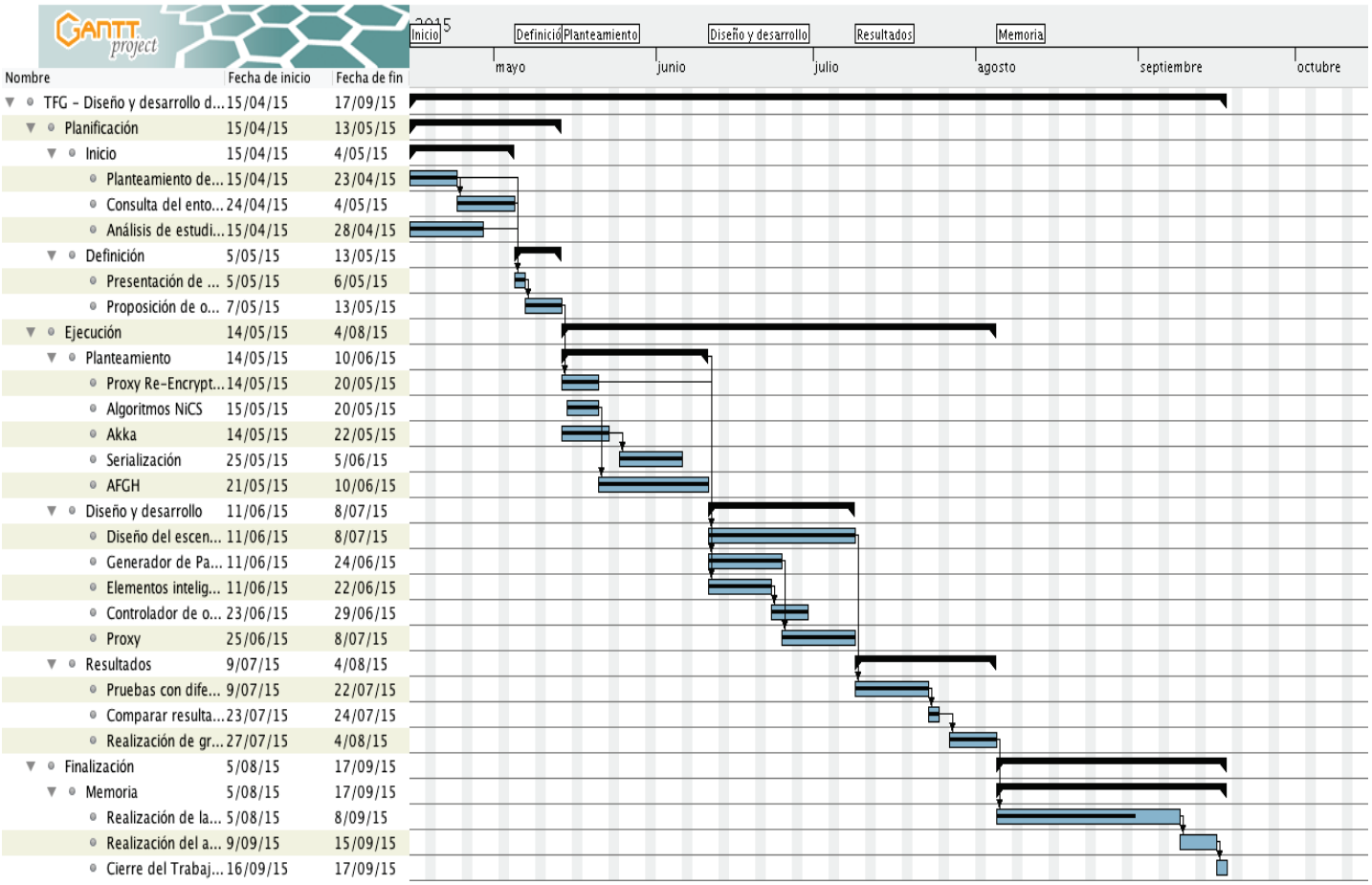


Figura 34. Diagrama de Gantt del proyecto presentado en este TFG.

Se puede observar en el diagrama de gantt las fechas de inicio y fin de cada tarea. Este trabajo se comenzó el 15 de abril de 2015, comenzando por la etapa de **Planificación**, que contiene las siguientes partes:

Planificación: Desde el 15 de abril hasta el 13 de mayo de 2015.

- Inicio: Desde el 15 de abril de 2015 hasta el 4 de mayo de 2015.
  - *Planteamiento del problema y fijación de objetivos.*
  - Consulta del entorno de estudio.
  - Análisis de estudios previos relacionados.
- Definición: Desde el 5 de mayo de 2015 hasta el 13 de mayo de 2015.
  - *Presentación de soluciones al tutor.*
  - *Proposición de objetivos.*

La segunda etapa del proyecto consiste en la **Ejecución**, es la etapa más extensa del Trabajo Fin de Grado, contiene las siguientes partes claramente diferenciadas:

Ejecución: Desde el 14 de mayo de 2015 hasta el 4 de agosto de 2015.

- Planteamiento: Del 14 de mayo de 2015 al 10 de junio de 2015.
  - *Proxy Re-Encryption.*
  - Algoritmos propuestos en la librería NiCS.
  - La librería Akka.
  - Serialización.
  - *El algoritmo unidireccional AFGH.*
- Diseño y Desarrollo: Del 11 de junio de 2015 al 8 de julio de 2015.
  - *Diseño del escenario de pruebas.*
  - Bloque generador de parámetros y pares de claves.
  - Bloque de actores orígenes, elementos inteligentes, sensor.
  - Bloque de actor destino, controlador de objetos.
  - Bloque de actor proxy, recifrador de mensajes.
- Resultados: Del 9 de julio de 2015 al 4 de agosto de 2015.
  - *Pruebas con diferentes configuraciones de tamaño de clave.*
  - *Comparar resultados entre las distintas configuraciones.*
  - *Realización de los gráficos de muestra.*

La tercera y última de etapa consiste en la Finalización, el cual incluye la memoria y entrega del Trabajo Fin de Grado:

Finalización: Desde el 5 de agosto de 2015 hasta el 16 de septiembre de 2015.

- Memoria y entrega del Trabajo de Fin de Grado:
  - *Realización de la Memoria.*
  - *Realización del anexo obligatorio en la lengua inglesa.*
  - *Cierre del Trabajo Fin de Grado.*

Se pueden observar, en cursiva, las tareas críticas de este proyecto.

AP.3: PRESUPUESTOS

En este apartado se va a proceder a desglosar los costes que generará la ejecución de este proyecto, para ello nos vamos inicialmente a basar en las tareas del proyecto definidas en el apartado anterior AP.3:

- **Inicio:** Del 15 de abril al 4 de mayo de 2015.
- **Definición:** Del 5 de mayo al 13 de mayo de 2015.
- **Planteamiento:** Del 14 de mayo al 10 de junio de 2015.
- **Diseño y Desarrollo:** Del 11 de junio al 8 de julio de 2015.
- **Resultados:** Del 9 de julio al 4 de agosto de 2015.
- **Finalización:** Del 5 de agosto al 16 de septiembre de 2015.

Teniendo en cuenta únicamente los días lectivos, definimos el número de días de trabajo tal y como mostramos a continuación (ver el Diagrama de Gantt de la Figura 34 para obtener más información sobre las tareas de cada parte del proyecto):

- **Inicio:** 13 días x 4h/día.
- **Definición:** 07 días x 4h/día.
- **Planteamiento:** 21 días x 4h/día.
- **Diseño y Desarrollo:** 20 días x 4h/día.
- **Resultados:** 19 días x 4h/día.
- **Finalización:** 31 días x 4h/día.

  - **TOTAL:** **111 días lectivos = 444 horas de trabajo**

➢ **COSTES DEL PERSONAL:**

Para calcular los **costes de personal**, tenemos en cuenta la siguiente tabla, la cual incluye tanto al autor de este Trabajo Fin de Grado como al doctor Tutor (se considera aproximadamente que el tutor participa en el proyecto un 10% de las horas lectivas del Graduado):

CATEGORÍA	TIEMPO (en horas)	PRECIO (€/h)	TOTAL (en Euros)
Dr. Ingeniero (Tutor)	45 h	60 € / h	2.700 €
Graduado	444 h	30 € / h	13.320 €
COSTES DE PERSONAL - TOTAL:			**16.020 €**

Tabla 31. Proyecto. Costes del personal.

➢ COSTES DEL MATERIAL:

Para el cálculo de los **costes del material**, tenemos en cuenta tanto el ordenador con capacidad de virtualizar una máquina virtual, las licencias correspondientes al software y todos los recursos necesarios tanto para el funcionamiento de los equipos (electricidad) como para realizar la labor de estudio del tema (conexión a Internet y a páginas relacionadas con la investigación), por ello tomamos los siguientes conceptos en la siguiente tabla:

CONCEPTO	PRECIO
Ordenador x64 con tecnología Intel-VT	750 €
Licencia Windows 10 Pro	279 €
Licencia Microsoft Office Professional 2013	350 €
Licencia Microsoft Project Standard 2013	500 €
Licencia Adobe Acrobat Pro DC (6 meses)	188 €
Gastos de Electricidad a la compañía eléctrica	80 €
Conexión a Internet Movistar FTTH (6 meses)	300 €
COSTES DEL MATERIAL - TOTAL:	**2.447 €**

Tabla 32. Proyecto. Costes del material.

➢ COSTE TOTAL DEL PROYECTO:

Por último, realizamos un **cálculo total del coste del proyecto incluyendo impuestos** en la siguiente tabla:

CONCEPTO	PRECIO
Costes de Personal	16.020 €
Costes de Material	2.447 €
TOTAL COSTES:	**18.467 €**
Impuestos (IVA 21%)	3.878,07 €
COSTE TOTAL DEL PROYECTO:	**22.345,07 €**

Tabla 33. Proyecto. Costes totales.

ENLACES, REFERENCIAS Y BIBLIOGRAFÍA

[1] "Broadband by the numbers. An Internet built for all; Fast, affordable and competitive". 2015. https://www.ncta.com/broadband-by-the-numbers

[2] "Behind the numbers: Growth in the Internet of Things". 2015. https://www.ncta.com/platform/broadband-internet/behind-the-numbers-growth-in-the-internet-of-things/

[3] "FlowCloud", Cloud solutions for Security, health monitoring, energy management and content delivery. 2015. http://imgtec.com/flowcloud/

[4] Nunez, D.; Agudo, I.; Lopez, J., "Integrating OpenID with proxy re-encryption to enhance privacy in cloud-based identity services," in Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on , vol., no., pp.241-248, 3-6 Dec. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6427551&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D6427551

[5] "JPBC: The Java Pairing Based Cryptography" library. 2015. http://gas.dia.unisa.it/projects/jpbc/faq.html

[6] IoT – What is the Internet of Things? – Definition of IoT. 2015 https://www.techopedia.com/definition/28247/internet-of-things-iot

[7] IoT – Ilustración descriptiva sobre la Internet of Things. 2015. http://www.eliassierra.com/negocios-online/hablemos-de-el-internet-de-las-cosas-internet-things/

[8] "Google Self-Driving Car Project". 2015. http://www.google.com/selfdrivingcar/

[9] "Google Self-Driving Car Project Components". 2015. http://nextpremium.com/2014/11/googles-self-driving-car-explained-one-infographic/

[10] Kevin Ashton, "That 'Internet of Things' thing", in RFID Journal , 22 Jun. 2009. http://www.rfidjournal.com/articles/pdf?4986 https://es.wikipedia.org/wiki/Internet_de_las_cosas

[11] Huadong Ma; Dong Zhao; Peiyan Yuan, "Opportunities in mobile crowd sensing," in Communications Magazine, IEEE , vol.52, no.8, pp.29-35, Aug. 2014. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6871666&isnumber=6871655

[12] Manejar la Concurrencia con Actores. 2014. http://www.genbetadev.com/paradigmas-de-programacion/manejar-la-concurrencia-con-actores

[13] Hewitt K. "Actor Model of Computation: Scalable Robust Information Systems", Arxiv.org, 1973. http://arxiv.org/pdf/1008.1459.pdf

[14] Definition of 'Actor Model'. 2015. https://en.wikipedia.org/wiki/Actor_model

[15] "The Reactive Manifesto". 2014. http://www.reactivemanifesto.org/es

[16] JAVA – FAQ - Qué es y para qué sirve. 2015. https://www.java.com/es/download/faq/whatis_java.xml

[17] JDK – FAQ - Qué es y para qué sirve. 2015. https://www.java.com/es/download/faq/develop.xml

[18] What is Akka? – Akka library for Java and Scala. 2015. http://doc.akka.io/docs/akka/2.3.13/intro/what-is-akka.html

[19] Actor Systems in Akka. 2015. http://doc.akka.io/docs/akka/2.3.13/general/actor-systems.html#actor-systems

[20] Supervision and monitoring on Akka Actor Systems. 2015. http://doc.akka.io/docs/akka/2.3.13/general/supervision.html#supervision

[21] Actor references, paths and addresses. 2015. http://doc.akka.io/docs/akka/snapshot/general/addressing.html

[22] Serialization and Deserialization progress in Java Scenario. 2010. https://mariosimaremare.wordpress.com/2010/02/03/serialize-object-and-deserialize-in-javatm/

[23] The Kryo Serializer for Akka. 2015. https://github.com/romix/akka-kryo-serialization

[24] The Proxy Re-Encryption Library. 2015. http://spar.isi.jhu.edu/prl/

[25] Giuseppe Ateniese; Kevin Fu; Matthew Green; Susan Hohenberger, "Improved Proxy Re-Encryption Schemes with Applications to Secure Distributed Storage". 2015. http://spar.isi.jhu.edu/~mgreen/proxy.pdf

[26] Nunez, D.; Agudo, I.; Lopez, J., "Integrating OpenID with proxy re-encryption to enhance privacy in cloud-based identity services," in Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on , vol., no., pp.241-248, 3-6 Dec.2012 http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=6427551&url=http%3A%2F%2Fieeexplore.ieee.org%2Fxpls%2Fabs_all.jsp%3Farnumber%3D6427551

[27] The JPBC Library for Java. 2015. http://gas.dia.unisa.it/projects/jpbc

[28] The NiCS Library for Java. 2015. https://www.nics.uma.es/dnunez/nics-crypto

[29] Susan Hohenberger; Zachary Scott, "Special Topics in Theoretical Cryptography", 2 Apr.2007. http://www.cs.jhu.edu/~susan/600.641/scribes/lecture17.pdf

[30] Solinas Prime Numbers. Definition. 2015. http://www.mathworks.com/matlabcentral/fileexchange/52365-solinas-prime